Requirements Engineering
https://doi.org/10.1007/s00766-021-00362-4

ORIGINAL ARTICLE

A use case driven approach to game modeling

Aghyad Albaghajati[1] · Jameleddine Hassine[1]

Received: 20 July 2020 / Accepted: 17 August 2021

Abstract
With the increase in market needs, game development teams are facing a high demand of creating new games every year. Although several methodologies and tools were introduced to support the game development life cycle, there is still a lack of evidence that these techniques improve game requirements understandability among development teams. The use of models in requirements engineering is considered a promising approach to support requirements elicitation, negotiation, validation, and management. In the context of game development, game designers argue that models are hard to learn and would restrict their creativity. In this paper, we introduce a novel use case-based game modeling approach that extends the standard UML use case diagram. The proposed technique allows for better representation of game-related requirements and promotes a common understanding of game requirements among game development teams. Our approach is implemented in a tool, called game use case modeling, and its applicability is demonstrated using four well-known games, Super Mario Bros, Tetris, Just Dance, and The Walking Dead. Moreover, in order to assess the perceived understandability, learnability, and usefulness of the proposed approach, we have conducted a survey involving 29 participants from the game development community. Results indicate a very satisfactory agreement regarding the added value of the proposed approach and a willingness of adoption by the game development community.

Keywords Requirement engineering · UML use case diagram · Game development · Game requirements · Game use case modeling

1 Introduction

According to a recent study by gamesindustry.biz [1], the total revenue raised in the global game market of games (published on computers, consoles, mobile phones, and the web) in 2018 was about 134.9 billion US dollars. Moreover, the game market is foreseen to grow tremendously in the next few years to reach a market value of 175.9 billion US dollars in 2025 [2]. Despite this continuous growth in sales and revenues, the game development process is known to be complex for many reasons [3]. One of the main reasons of such complexity is due to the fact that game creation involves several teams (supplying different game components) from different backgrounds, making the communication between teams difficult and unproductive [4–6]. In fact, game development differs from enterprise software development in two aspects. First, an enterprise software development process starts with the elicitation of stakeholders' goals and needs, whereas most of game development projects start with a story or a game concept established by the game development team. Second, game development targets a mass market of customers having, generally, different backgrounds and interests [7].

According to Reyno and Cubel [8], many attempts have been made to introduce software development methodologies in game development. For example, the adoption of the waterfall development model in game development pushed development teams to scale up their efforts to produce extensive natural language documentation [9]. However, maintaining such large specifications is a difficult and error prone activity, which may cause potential communication problems between development teams; hence leading to low productivity [9].

✉ Jameleddine Hassine
jhassine@kfupm.edu.sa

Aghyad Albaghajati
g201703510@kfupm.edu.sa

1 Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran 31261, Kingdom of Saudi Arabia

Agile methodologies help cope with many of these risks by promoting the communication and interaction between team members [10]. In addition, agile methodologies enhance change management during game development [11] by emphasizing the implementation of the core mechanics of the game play[1] as fast as possible [13]. However, although the iterative nature of an agile process leads to shorter game development cycles and more frequent releases, it may shift the focus of development teams from the conceptual abstraction to programming details [14]. Consequently, development teams may lose insight and not see the game's "global picture." Moreover, according to a survey by McKenzie et al. [15], although many studios claimed that they are applying agile methodology in their gaming projects, they are in reality not following the key agile practices as intended. Godoy and Barbosa [16] proposed a game development methodology, called Game-Scrum, that is based on Scrum, an agile methodology that utilizes short iterations, called *sprints*. Game-Scrum [16] has three main stages, namely pre-production, production, and post-production [17]. The goal of the pre-production stage is to create and establish the game concepts and requirements related to the game's story, mechanics, and design. Moreover, game prototypes are created during this stage. The production stage focuses on creating and validating game assets such as visuals, animations, and software components. The post-production stage manages the advertisement, feedback, maintenance, and updates of the game after distribution [17]. The applicability of Game-Scrum was shown using a game for teaching software engineering. Although Game-Scrum seems promising, Godoy and Barbosa [16] stated that more studies and applications following this methodology are required and that their methodology is still not mature enough.

Moreover, a study by Folmer [18] has presented the use of component-based development methodology in game development in order to enhance the reusability of system's functionalities. However, the study has shown many limitations, such as the high amount of time and effort required to understand and integrate the developed components. Thus, there is no tangible guarantee that this methodology will speed up the development or enhance the reuse of functionalities.

Model-based development (MBD) [19] is a development methodology that relies on the use of models as primary development artifacts [7, 20]. Applying model-based development methodology to game development helps enhance the level of abstraction within the development life cycle which allows the teams to cope with the increasing complexity of game development process. Moreover, it improves the teams' productivity by conceptualizing game specifications before the implementation phase [8]. Furthermore, applying MBD techniques to game development would enhance the overall flexibility and efficiency of the development process and allows for a better fluidity during the game construction [21]. Thus, it would help enhance the communication and resolve the lack of documentation, considered as two of the top 10 most occurring problems in a game development project [22]. However, some game designers are often reluctant to adopt models in their work. They assert that these models are not suitable for designing games and would restrict creativity [21, 23].

Requirements engineering (RE) is widely considered as a critical activity in game development life cycle [4, 5, 22, 24, 25]. Callele et al. [4] have emphasized the importance of requirements engineering activities in supporting the creative process of game development. Later, in a survey by Petrillo et al. [22], the authors stated that the unrealistic estimation of scope is one of the main problems faced in game development process. This issue is mainly caused by a lack of extensive requirements analysis [22]. Software requirements engineering process consists of several tasks, such as requirements elicitation, modeling, analysis, and validation. Requirements models, expressed in various modeling notations, play a central role in the requirements elicitation phase [25]. Different requirements models are used to describe different aspects of the system, e.g., contextual, behavioral, non-functional, data management, etc. [25].

Recognizing the need for game-centric requirements engineering approach, we make the following contributions:

1. We introduce a novel game-oriented use case modeling technique, as an extension of the UML use case diagram [26]. The goal of this extension is to provide a simple, general, and practical game-related requirements modeling technique that may be used for modeling different game genres [27]. Furthermore, the proposed technique aims to streamline use case-based modeling for game development communities, which makes such techniques more accessible and usable.
2. As a proof of concept, we have developed a tool, called *game use case modeling (GUCM)*, that implements our proposed use case-based game modeling technique.
3. We demonstrate the applicability of our approach and tool by applying it to four well-known commercial games, namely Super Mario Bros [28], Tetris [29], Just Dance [30], and The Walking Dead [31].
4. We have evaluated the perceived learnability, understandability, and usefulness of our proposed approach, by conducting a survey involving 29 participants from the game development community. Results indicate the added value of the approach.

[1] Djaouti et al. [12] defined game play as a central element within a video game defining its quality in the mind of the players.

It is worth noting that the proposed use case-based technique is mainly designed to be used during the game requirements elicitation stage. However, it can be adapted and integrated within a game based MBD process (using model transformations and code generation), a waterfall process, or even within an agile modeling (AM) methodology [32, 33]. Indeed, in agile modeling (AM), requirements can be modeled using use case diagrams for better understanding of the system to be [33] and to aid communicate requirements effectively [32].

The remainder of this paper is organized as follows. Section 2 provides the necessary background of this research. Related work is presented in Sect. 3. Section 4 describes our proposed use case-based game modeling approach. Our GUCM prototype tool is presented in Sect. 5. The applicability of the proposed approach and tool is shown using four illustrative examples in Sect. 6. The empirical validation of our approach is presented and discussed in Sect. 7. Threats to validity are discussed in Sect. 8. Finally, Sect. 9 concludes and refers to future research directions.

2 Background

In this section, we first provide a brief introduction to UML use case model [26] main components, and then we discuss the roles of team members involved in game development.

2.1 UML use case diagrams (UCD)

The Unified Modeling Language (UML) [26] is a graphical modeling language used to specify and document the artifacts of a software system [26]. UML provides a wide variety of sublanguages, each of which provides different description capabilities and conveys different information, e.g., structural, behavioral, etc. One of the most commonly used UML diagrams is the use case diagram (UCD) [34–36]. A use case diagram is a graphical depiction of a use case model in UML [37]. It is used to capture functional system requirements and is composed of four types of components: (1) *actor* (represented graphically as a "stick man") specifies a role played by a user, an organization, or an external system that interacts with the system, (2) *use case* (represented graphically as an oval) specifies a unit of useful functionality that the subject provides to its users [26], (3) *subject* (also called *system boundary*, depicted as a rectangle) represents the system under consideration [26]. Use cases are located within the subject, and (4) *relationship* (represented graphically as a directed or undirected link) that can be of four types: (1) *association* used between actors and use cases to capture that an actor participates in the use case (2) *include* used between use cases to extract common behavior of two or more use cases or to reduce the complexity of a use case, (3) *extend* used between use cases to specify an optional or exceptional behavior that can be introduced according to some predefined extension points/constraints, (4) *generalization* is used between actors to specify that an actor (called *concrete actor*) can inherit the role of another actor (called *abstract* actor), and between use cases to state that a use case is defined as a specialized form (i.e., *concrete use case*) of an existing use case (*abstract use case*).

The graphical representation of a use case diagram may be complemented by a structured textual specification. A structured textual use case provides a detailed description of the use case [38]. A use case template may include the use case name, number, goal, description, primary actors, secondary actors, offstage actors, special requirements, preconditions, post-conditions, priority, frequency, main flow, sub-flows, alternative flows, extension points, exceptions, super use case, sub use case, due date, and open issues [38].

UML provides two techniques for extending its base meta-model to support domain specific modeling: (1) A UML profile is a predefined set of stereotypes, tagged values, and constraints. A stereotype defines how an existing meta-class may be extended. A stereotype is specified between double angle brackets, i.e., ≪stereotype≫, (2) adding new meta-classes along with metadata and meta-associations. Such extensions can be implemented using the Meta-Object Facility (MOF) [39], OMG meta-modeling and metadata repository standard.

2.2 Game development teams

In order for a game project to succeed, game development teams must work together in a very coordinated and consistent process. Game development requires a very skilled team that consists of many members where each member has his own role and a specific task to accomplish. Effective communication between game development team members is considered as one of the most essential project success factors [40].

There are four main roles that support the game development process which can be described as follows:

- *Game designer* Game Designer is responsible for the creation of game play, game rules, and game structure. The game design team may participate in designing the user experiences, user interfaces, game documentation, narration, and content. Moreover, the creation of game characters, their graphics, their roles, their voices, and their visuals could be performed by the game designer. The game design team may be organized around several members, each working on a specific game design aspect [41].
- *Artist* Game art plays an important role in game development. A game art, created by an artist, includes creat-

ing game environments, game models, characters, game visual effects, and game animations for both characters and environmental objects, and for both two- and three-dimensional games [41].

- *Level designer* Game levels are designed by a "level designer" who is responsible for creating level challenges, difficulties, and stages in a game. A level designer helps create the game play components and write game stories that are related to the game's mission and objectives [41].
- *Game programmer* Game programmer creates the mechanics of a game. He is generally involved in the creation of game engine, databases, graphics, audio, and any game development tools that support other team members during the game production process [41].

In addition to these four main roles, there are other roles that can help manage and produce high quality games, such as the roles of managers, game producers, quality assurance, and testing teams.

3 Related work

In this section, we survey related work from two perspectives: (1) studies that use models during the various phases of the game development life cycle and (2) studies related to extending UML use cases diagrams (UCD) to serve specific domains and contexts.

3.1 Use of models in game development

Tang et al. [42] presented a conceptual modeling framework that utilizes UML class diagrams, in addition to the use of a modified state transition diagrams to represent game components, screens, and the flow of game interactions and events. The authors [42] stated that their framework aims to define a high-level abstraction of serious game applications and would support the separation between low-level activities, such as coding and game assets, from design tasks. However, their approach presents technical details that require software development background, making it less suitable for non-software engineering teams, hence affecting the communication between game development software and non-software teams. Our proposed technique is mainly designed to be used during the requirements elicitation stage and uses use case diagrams to enhance game requirements understandability among stakeholders. Other design aspects, such as entity attributes and actions (usually described using class diagrams), may be addressed in later development stages.

Game technology model (GTM), introduced in [43], is a game modeling technique for serious games (i.e., games that are based on role-playing and simulation game genres). GTM is based on a data-driven architecture and is independent of any hardware or operating platform. The authors [43] presented a web-based modeling tool called *SeGMEnt* that was developed to allow non-technical domain experts to document serious games. The modeling tool supports various design viewpoints, namely object, simulation, structure, presentation, player, and environment. One of the limitations of their approach is its lack of details about game objects and their associated events, their roles, and their artistic representations, like animations and visuals. Indeed, the presented modeling approach focuses on the data flow between game states, which may limit the understanding of the game context and may hinder the communication between team members, especially between developers and artists. Moreover, GTM fails to include information about preconditions or post-conditions of states and events.

Sauer and Engels [36] introduced a modeling language for multimedia applications called *Object-oriented Modeling of MultiMedia Applications (OMMMA)*. OMMMA is based on extending various UML diagrams (i.e., objects, sequence, and state charts diagrams) to model temporal (i.e., time point or time interval relations) requirements of multimedia applications. The authors [36] claimed that their technique supports the transition from traditional application models to multimedia application models without the need to learn new modeling paradigms or incurring the hassles of language shifts. However, the proposed extensions (to several UML sublanguages) might be daunting and complex especially for team members who do not have a software engineering background [44]. In our approach, we focus only on describing game requirements at the use case level. Aspects like temporal dependencies are left to the design stage, where designers are free to choose any method they think suitable.

In another study, Hernandez and Ortega [45] introduced a graphical domain-specific language for modeling 2D video games, called *Eberos GML2D*. The authors [45] claim that the use of their language would reduce considerably the complexity of game development life cycle. The proposed language consists of several constructs, representing various game components. These constructs are used to describe: (1) the *sprites* containing graphical elements and animations, (2) the *entities* representing player or non-player characters, (3) the *logic* of the game entities, (4) the *controllers* that allow the modeling of game managers, and (5) the detection of *collisions*. Despite the variety of constructs exhibited by the modeling approach, no textual description was considered, which may limit the ability of team members to add extra details. In our proposed approach, we extend the UC diagram with game-related stereotypes and we consider both graphical and textual representations, allowing for more flexibility of game requirements descriptions.

Herzig et al. [46] introduced a formal language for modeling gamification concepts, called *Gamification Modeling Language (GaML)*. GaML is designed to help decouple specification and design phases from the implementation phase, allowing for the validation of game mechanics and gamification concepts. GaML focuses mainly on depicting the flow of events in a game. Moreover, it supports object constraint language (OCL) to add constraints and conditions to the events. The authors [46] claim that the models produced using GaML can be read by both technical and non-technical development team members. However, GaML does not capture details about some basic game building blocks like game objects, animations, or scenes. Furthermore, GaML does not support the textual description of game events, a feature that may be required to achieve a better understandability. Our game UC-based approach provides a textual representation of the modeled use cases and covers all game-related aspects, e.g., load scene, play animation, play audio, create objects, and functions.

Reyno and Cubel [47] proposed a game modeling approach that is based on three main diagrams: (1) a structural diagram represented by a UML class diagram with stereotypes to capture the structure of the modeled game, (2) a behavioral diagram represented by a UML state diagram that describes the behavior of the defined entities, and (3) a control diagram represented by an object diagram that maps actions to controls. The authors [47] stated that their modeling approach helped in increasing the team productivity while reducing the development time. Our proposed approach focuses on game requirements modeling using use cases rather than the detailed game design using structural and refined behavioral models. However, our approach may be considered as a good fit to fill the requirements elicitation gap in the work of Reyno and Cubel [47].

Pleuss and Hussmann [48] presented a model-driven development approach which integrates software design, user interfaces, and media into a single consistent modeling language. The models serve as a contract between the different teams working on the project. The modeling language discussed in the study [48], called *Multimedia Modeling Language (MML)*, supports advanced multimedia integration modeling. Models created by this language include: (1) task model that reflects the user tasks supported by the application, (2) structure model that represents domain classes of the application logic such as media classes (video, audio, or animation) which act as traditional UML class diagrams, (3) scene model which shows the state of the application and its association with the user interfaces which is modeled using UML state chart diagram, (4) presentation model which specifies each scene's user interface and contains the relationships between scenes, instances of domain classes from structure model and presentations, and (5) interaction model which is represented by UML activity diagrams and shows interactions of the user with the game scene; in addition, it depicts relationships between presentations, scenes, tasks, and instances of the domain classes. However, although the authors [48] claimed that their modeling approach will serve as contract between different teams, having multiple graphical representations may cause understanding and communication issues between the stakeholders. Moreover, most of the used models require software engineering development background. We believe that our proposed game use case diagram may be integrated within MML [48]. However, this is out of the scope of this paper and will be addressed in future work.

Lope et al. [49] proposed the use of UML diagrams to model educational games. The authors [49] suggested that class diagrams can be used to model game structure, state diagrams to model scenes, acts, and scenarios, activity diagrams to model actions, and sequence diagrams to model challenges and details of actions. However, focusing mainly on actions and scenes might prevent the models from capturing other game details, such as audio, animation, game objects, and characters. On the other hand, the use of different types of models would increase the complexity of the project, limiting the understandability and communication, especially when a team includes members with limited background in UML modeling. Our proposed approach focuses mainly on game-oriented use case modeling (describing characters and events/functions), in addition to its ability to capture details related to scenes, game objects, animation, and audio.

Considering the surveyed studies, we can conclude that there is a research gap with respect to the use of modeling techniques to support game-related requirements understandability and establishing a common context of communication within heterogeneous game development teams (composed of technical and non-technical members). Indeed, most of the surveyed studies focused on describing the flow of events, game architectural design, and code generation aspects and little attention was devoted to the creation of models to facilitate the involvement of non-technical team members in the game development life cycle. Moreover, several studies were found to propose modeling approaches that depend on different types of models, which might not help improve requirements understandability between development teams. Furthermore, most of the surveyed studies targeted specific types of games such as serious and educational games, which limits the generality of the proposed techniques. In order to fill this gap, we propose a general and simple modeling technique based on UML use case diagram that can be used to describe game requirements of different game genres [27]. Furthermore, our proposed approach would help establish a common understanding ground about the game requirements among the development teams.

3.2 UML use case extensions

Several UML use case extensions have been proposed in the literature in order to adopt use case modeling to different situations and contexts [50–54]. Yu et al. [53] presented a UML use case extension, called AspectRUCM, inspired from the Aspect-Oriented Requirements Engineering. It has been developed to support the needs of the industry when dealing with specific crosscutting concerns. Their technique [53] uses stereotypes in both actors and use cases to specify context specific types. Moreover, their proposed approach introduced new relationships, such as "trigger" which indicates the interaction with a use case. Hog et al. [50] have proposed a modeling solution for adaptive web services, based on the UML use case diagram. The authors [50] have used stereotypes to categorize and classify use cases. Furthermore, various types of actors were introduced to represent entities interacting with the web service, where each actor is identified by a name and specific icon, e.g., provider, human consumer, application consumer, and composite web service. Murali et al. [51] introduced a safety-oriented requirements engineering modeling technique. The authors [51] have illustrated several extensions such as safety specific use cases (e.g., accident scenarios) and safety-specific relationships like "mitigate" and "disrupt."

Al-Alshuhai and Siewe [52] have proposed an extension to UML use case diagrams to model context-aware applications, called *context-aware use case Diagram*, where both context and behavioral aspects have been addressed using new graphical elements. Moreover, a new relationship, called "utilize," was introduced between a use case and a context use case, a newly defined use cases type.

In Mai et al. [54], the authors have presented a use case technique that extends UML use cases to model security requirements. Misuse cases were defined in the study as a sequence of events performed by a malicious actor to cause harm. Misuse cases were drawn in gray to distinguish them from normal use cases. The authors used the ≪security≫ stereotype for security use cases that describe countermeasures against misuse cases. The textual description of the use cases was extended with a template and restriction rules that allow for precise description of the misuse case. In addition, the misuse case post-condition field specifies which assets are potentially impacted. Moreover, the authors [54] extended the basic and alternative flows to create Basic Threat Flow, Specific/Bounded/Global Alternative Flow and Specific/Bounded/Global Alternative Threat Flow. These extensions have brought two new use case relationships, other than includes and extends, namely "mitigate" and "threaten." In addition to the use case extensions, actors were also extended to describe malicious actors/external systems [54]. Malicious actors were labeled as "malicious." Their approach [54] is implemented in a tool that checks the consistency between the specification and the misuse case diagram.

Cooper and Longstreet [55] presented a modeling approach for serious educational games using an extended UML use case diagram. This approach is based on two components, the use cases and a tabular specification. The use case extension presented in the study can be distinguished from regular ones using stereotypes, e.g., Game, Act, Scene, Screen, and Challenge. The associations used between the elements are "uses" and "includes." The tabular specification defines each stereotype used in the use cases, providing more details about the game play. Their proposed approach focuses on modeling scene-based educational games; hence, its applicability cannot be generalized. Moreover, the introduced stereotypes are limited to scene-based educational games, where there are no details about animations, game objects, or audio. In addition, the approach does not support ≪extends≫) relationships nor extension points. Furthermore, there is no (subject/system boundary) present in the resulted models.

4 A use case-driven approach to game modeling

In traditional UML use case diagrams, actors, use cases, and the relationships between them are used to represent the functionalities of a software system (as described in Sect. 2.1). UML use case diagrams have been proven to be very useful, in enterprise and traditional software development processes [56], due to their abstract aspect, allowing for a better understanding of system's context and functionalities among stakeholders, domain experts, and other professionals with diverse backgrounds. However, in the game development field, there is still a lack of support of use case models, due to the fact that (1) not all game development teams have a software engineering background, and (2) use case diagrams do not capture precisely the game context and its internal behaviors (in the language used by game development teams), which might affect the communication and understanding of the project.

In order to help cope with these issues in the game development context, we propose a use case-based game modeling approach that aims to provide a general use case language that captures game context and behavior and that can be applied to many game genres [27], e.g., *Action*, *Fighting*, *Adventure*, etc. Moreover, the proposed technique is designed to be used mainly during the game requirements elicitation stage. However, given the iterative nature of game development processes [40, 57], where game features are built iteratively, leading to regular and frequent updates, the game-oriented use case models may be updated during the post-production phase. Furthermore, we aim to make the

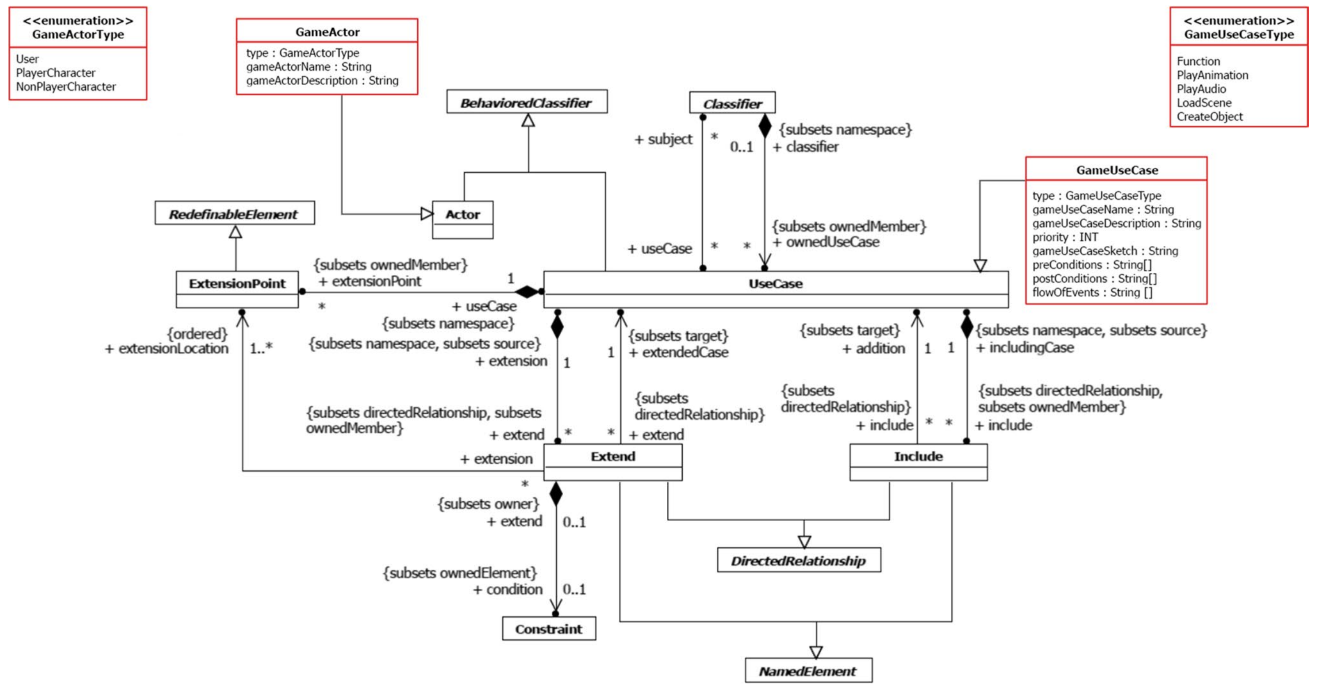


Fig. 1 Metamodel of the game use case diagram—extended from UML 2.5.1 use case diagram [26]

adoption of use case-based modeling within game development communities, more accessible and more usable.

In what follows, we first present our proposed use case meta-model extensions along with the corresponding textual descriptions. Next, we present our game-related syntactic rules that aim to produce well-formed use cases.

4.1 Extending UML use case meta-model for game requirements

Figure 1 illustrates the UML use case meta-model abstract syntax augmented with our game-related extensions. The standard UML use case meta-classes are shown in black, while the four meta-classes in red boxes represent the newly proposed meta-class extensions:

1. *GameUseCase* to represent the new game use cases.
2. *GameUseCaseType* that is an enumeration type enclosing the following types *Function*, *PlayAnimation*, *PlayAudio*, *LoadScene*, or *CreateObject*.
3. *GameActor* to allow for the description of different types of actors.
4. *GameActorType* is an enumeration that can have one of the following three values *User*, *PlayerCharacter*, and *NonPlayerCharacter*.

In the following subsections, we describe in details our proposed UC extensions.

4.1.1 Game actors

Actors in traditional UML use case diagrams are used to model the roles played by entities (external to the system) that interact (e.g., through data exchange) with the system. In our proposed technique, we extend the notion of actors to represent game characters, game objects that are controlled by the player, and any other game entity that has a standalone behavior. Each actor is identified by a name and a graphical representation.

We consider the following three types of actors:

- *User* Used to represent users of the game, i.e., players. We adopt the standard UML use case stick-man representation to represent game users.
- *PlayerCharacter* Used to represent game characters and game objects that are controlled by players. For example, *Mario* in *Super Mario Bros* [28], *Sonic* in *Sonic the Hedgehog* [58], *Tetris Pieces game objects* in *Tetris* game [29], etc. The graphical representation of a PlayerCharacter actor is stick-man and a joystick icon next to it, which denotes a game object being controlled by the player.
- *NonPlayerCharacter* Used to represent characters controlled by the game system, e.g., enemies, characters, or any other game object that behaves and makes decisions on its own. The graphical element of *NonPlayerCharacter* actor is a stick-man and a computer icon next to it, which denotes a non-player character.

Table 1 Types and graphical representation of game-related actors

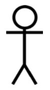
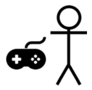
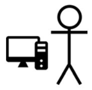
Actor type	Description	Graphical representation
User	Represents the users of the game, e.g., players, maintainers, developers, designers, artists, etc.	
PlayerCharacter	Represents game characters or game objects that are controlled by the players	
NonPlayerCharacter	Represents the characters or game objects controlled by the game system, e.g., enemies.	

Table 1 describes actor types, descriptions, and their graphical representation.

In addition to the graphical representation, we provide a textual description of the game actor. This would allow for a better understanding and documentation of the game actors composing the model. The textual representation is implemented in our prototype GUCM tool, described in Sect. 5. An actor has the following attributes:

- *Game actor name* denotes the actor name.
- *Game actor types* denote the actor type, e.g., *User*, *PlayerCharacter*, or *NonPlayerCharacter*, as described in the metamodel of Fig. 1.
- *Game actor description* provides a brief description of the actor.
- *Game actor use cases* lists all game use cases associated with the game actor.

4.1.2 Game use cases

A use case in traditional UML use case diagrams is used to represent a functionality or a behavior of the system. In our proposed technique, we extend the notion of use cases to represent a variety of game-related behavioral aspects, such as loading a game scene (described using *LoadScene* use case), playing an animation (described using *PlayAnimation* use case), etc.

We propose five new game-related stereotypes to distinguish game use cases:

- *Function* Used to describe functions or behaviors that are invoked by any type of the game use case's actors. An example of a function could be, start game, jump, attack, win, die, or other functions that the actor can perform depending on the game specification.
- *LoadScene* Used to describe the loading of scenes within the game. These scenes could be game levels, game main scene that contains the main menu, or other levels and scenes part of the game. The *LoadScene* game use case assumes that the scene components, e.g., graphics and environment components, have been already defined. Furthermore, it emphasizes that loading a specific scene/level is invoked after a certain behavior/event took place, or when a condition/constraint is satisfied. For example, the loading of the next level cannot take place unless the player wins the current level.
- *PlayAnimation* Used to describe animations played by game characters (either *PlayerCharacter* or *NonPlayerCharacter*). Animations are often played after certain events or may be included within some functions and behaviors. Moreover, in order to create the motion effect, animations are defined as a successive drawing of sprites or as changes of positioning of 2D/3D models of a game object. For example, the jump animation is triggered when a jump function is being executed.
- *PlayAudio* Used to describe audio playing. PlayAudio may be as general as playing a background music or it can be specific, i.e., a sound effect being played after a certain behavior, such as a jump sound effect.
- *CreateObject* Used to describe the instantiation of objects within the game. Objects may be enemies, projectiles, particles, 3D objects, or any other game entity that can be created. For example, gun bullets are created and fired to a specific direction, following the triggering of a firing function.

To distinguish game-related use cases from standard use cases, we use the stereotype feature, i.e., ≪Function≫, ≪PlayAudio≫, ≪PlayAnimation≫, ≪LoadScene≫, and ≪CreateObject≫.

Table 2 shows the different types (described using special graphical icons located on the top left of the use case oval) of game use cases.

Table 2 Types and graphical representations of game-related use cases

Function	LoadScene	PlayAnimation	PlayAudio	CreateObject
⚙	🎬	🏃	🔊	🧊

In addition, the textual representation of the proposed game use cases includes the following attributes:

- *Game use case name* denotes the game use case name.
- *Game use case description* provides a brief description of the game use case.
- *Game use case type* denotes the game use case type.
- *Game actors* denote game actors associated with this game use case.
- *Priority* This attribute was proposed by Cockburn [59], which represents how critical a use case is to the system and organization. In our proposed technique, we adopt a numerical value to measure this item, that ranges from 1 to 5, 1 being not critical, 5 being very critical.
- *Game use case sketch* A brief artistic representation or sketch, which describes the game use case visually.
- *Preconditions* denote preconditions that must be satisfied to execute the game use case.
- *Post-conditions* denote post-conditions that must be satisfied after the execution of the game use case.
- *Extension use cases* The use cases that extend the current game use case.
- *Included use cases* The use cases that the current use case includes.
- *Flow of events* describes the events flow of the game use case.

It is worth noting that preconditions and post-conditions can be expressed either as logical expressions or as natural language. We do not impose the use of a specific formalism, such as OCL, to minimize the learning curve for non-software engineering team members and to promote the adoption of the proposed technique.

4.1.3 Subject

The subject represents the game under development. It is depicted as a rectangle with its name in the corner (⌐). Actors of type "User" have to be located outside the subject boundary, while game use cases are located within the subject. Furthermore, we adopt the approaches presented in [60, 61] for representing software agents within UML 2.0 use case diagrams. Software agents, being part of the system under consideration, are placed within the system boundary [60, 61]. In our context, since actors *PlayerCharacter* and *NonPlayerCharacter* are part of the game, we place them inside the subject boundary.

4.1.4 Relationships

In our proposed technique, we do not propose any new relationships between game-related use cases. However, game-related use cases can still use standard UML use cases relationships, i.e., includes and extends, as described in Sect. 2.1. Extension points (represented as notes ◻) are used to show conditions/triggering events, when extending use cases.

4.2 Game-oriented use case well-formedness rules

In order to ensure the validity and well-formedness of the produced game-oriented use case models, we propose two sets of rules, namely strict and soft rules.

4.2.1 Strict rules

Strict rules are enforced by our GUCM prototype tool to make sure that the resulting model is valid. Breaking such rules would invalidate the game use case model. We have defined eleven rules:

- *Strict rule 1* An actor should be linked to at least one use case.
- *Strict rule 2* A use case should be connected to at least one actor or one use case.
- *Strict rule 3* An actor/use case must have a name. The name is used to identify the actor/use case; hence it should be unique.
- *Strict rule 4* A use case shall not be allowed to both include and extend the same use case.
- *Strict rule 5* A use case cannot include or extend itself.
- *Strict rule 6* Only one single extension point element is allowed per an *extends* relationship.
- *Strict rule 7* An extension point can only be associated with an *extends* relationship. It cannot be associated with an "actor-use case" or "includes" relationships.
- *Strict rule 8* An extension point shall not be shared among different *extends* relationships.
- *Strict rule 9* Game use cases should be placed within the subject boundary.
- *Strict rule 10* Actors of type "PlayerCharacter" and "nonPlayerCharacter" should be placed within the subject boundary.
- *Strict rule 11* Actors of type "User" should be placed outside the subject boundary.

4.2.2 Soft rules

Soft rules are recommended guidelines that modelers are encouraged to follow. They are designed mainly to avoid inconsistencies related to the different types of model relationships, although breaking these rules does not invalidate the produced models. Soft rules are classified into two categories: actor—use case and use case—use case relationships:

1. *Actor—use case relationships*
   - *Soft rule 1* A *User* actor can only be associated with a non-game use case, such as, start game, load level, save game, etc., since these use cases are not part of the game play.
2. *Use case—use case relationships*
   - *Soft rule 2 Function*, *PlayAudio* and *CreateObject* game-related use cases can include (i.e., ≪ includes≫) any type of game-related use cases except *LoadScene*. Indeed, the behavior of loading a scene is known to extend the flow of control of the game, which is triggered by another game-related behavior [62]. Thus, *LoadScene* shall not be included in other game use cases.
   - *Soft rule 3 PlayAnimation* and *LoadScene* game-related use cases cannot include (i.e., ≪includes≫) any type of game use cases. Indeed, game use cases *PlayAnimation* and *LoadScene* represent standalone behaviors and they are independent from other game uses cases. However, they support extensions to allow dynamic and flexible flow of control. On one hand, playing an animation involves changing sprites or objects' transformations (i.e., position, scale, or rotation) over time. However, this behavior can be extended to add audio, create objects, execute a function, or load a scene. On the other hand, the behavior corresponding to loading a scene is, in its basic form, a loading of game levels. In order to not disturb the loading, we shall not include other game use cases. However, it can be extended with animation or audio playing.
   - *SoftRule 4 LoadScene* game use case can extend (i.e., ≪extends≫) *Function* and *PlayAnimation* use cases only. On one hand, *Function* use cases can trigger the behavior of loading a scene, e.g., collecting a key, or entering through a door which can trigger loading a level. On the other hand, *PlayAnimation* game use cases can be extended by *LoadScene*, where a scene loading behavior gets triggered after certain animation is played. There are several types of animations in games [63], and one of these types is cut-scene animation, which is a cinematic sequence of animations. One such extension example is when a cut-scene animation is played at the end of a level to introduce the next one, which can be thought of as extending an animation with *LoadScene* behavior.
   - *SoftRule 5 PlayAudio*, *CreateObject* and *LoadScene* game use cases can be extended (i.e., ≪extends≫) by any type of game use cases except *LoadScene*. Loading a scene cannot be triggered by another *LoadScene* behavior, where there must be a *Function* use case that triggers the next *LoadScene*, or perhaps a *PlayAnimation* that triggers the next *LoadScene*. Moreover, *LoadScene* can be extended by other game use cases such as *Function*, where some levels might trigger the functionality of placing game objects in the level, or the functionality of saving player's progress while loading [64]. On the other hand, *PlayAudio* cannot be extended by a *LoadScene* game use case, because loading a scene means changing the flow of the game, and playing an audio does not necessarily support that action of flow changing. Moreover, we shall not extend a *CreateObject* use case with a *LoadScene* use case, because this would impact the game play. Indeed, loading a new scene while instantiating an object, may place the created object in the wrong scene, which would ruin the player's overall experience. Thus, to prevent such problems, one can only load scenes after certain function or animation.

5 Game use case modeling (GUCM) tool

Game use case modeling (GUCM) is a web-based modeling tool[2] that was developed to meet the use case-driven approach to game modeling and its objectives. The tool was built using Unity engine [65] and C# language. Unity uses a component-based approach to software development, where components are classes that inherit from *MonoBehaviour*, a Unity predefined base class. The GUCM tool is composed of 74 classes with a total of 9152 lines of code. Several of these classes inherit from Unity's *MonoBehaviour* class.

GUCM provides the following features:

1. *Graphical creation of the game use case model* GUCM uses a click and place technique to populate the game use case model with game actors, game use cases, extension points, and subjects (provided as part of a palette). Figure 2 shows GUCM's graphical elements creation palette. These graphical elements can be connected using unidirectional and bidirectional links. Figure 3

[2] Publicly available via https://mraghyad.github.io/GUCM/.

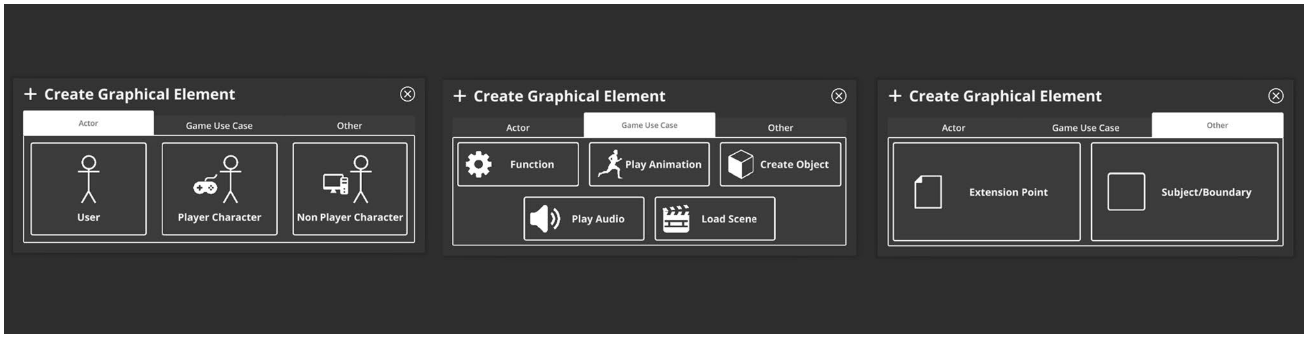


Fig. 2 GUCM tool's graphical element creation palette

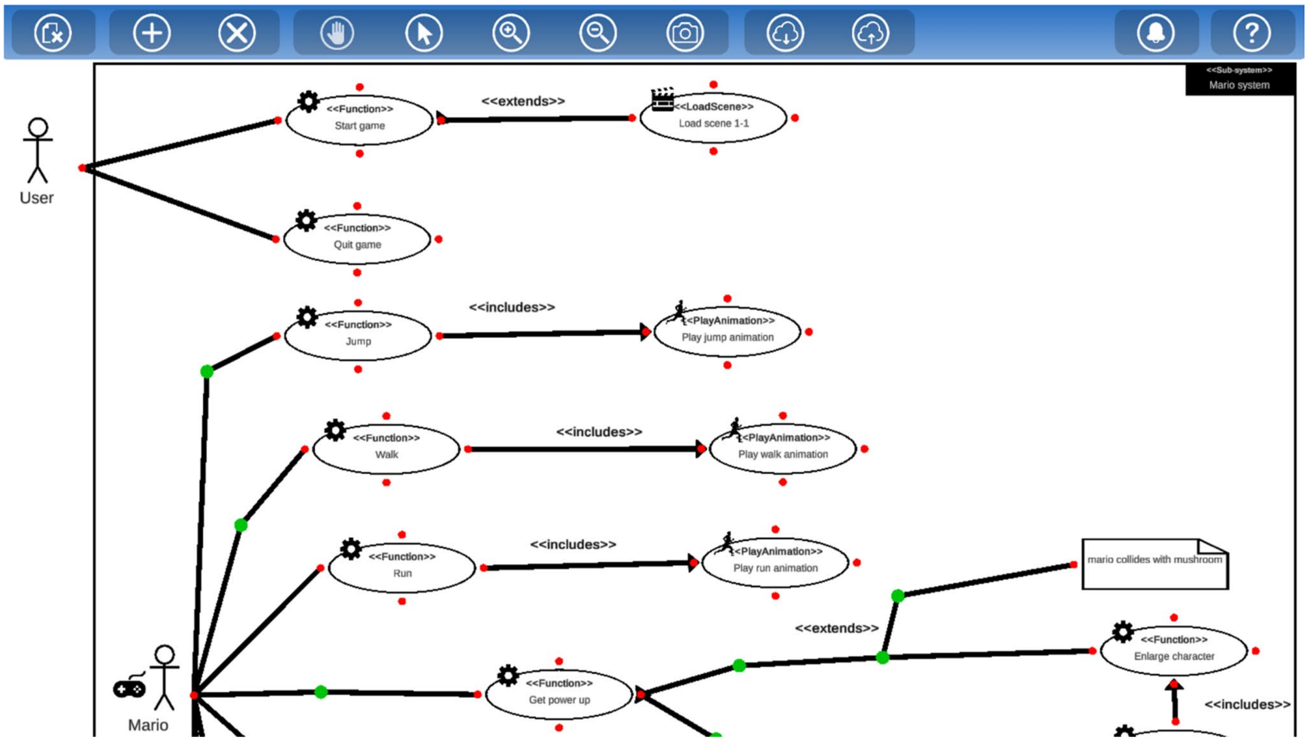
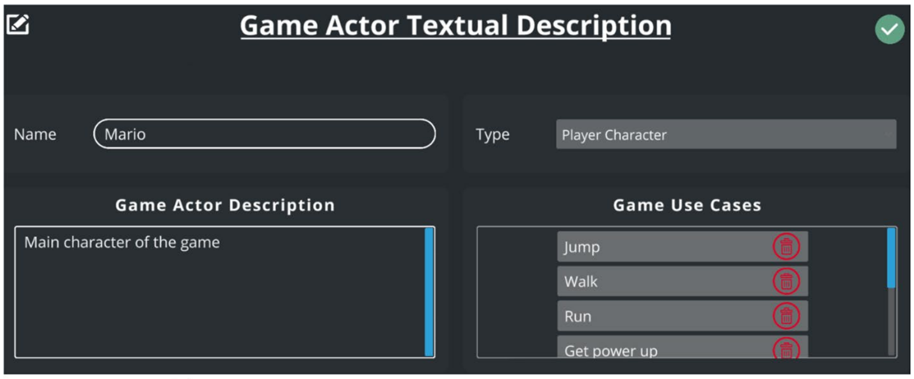

<<Sub-system>>
Mario system
User
Mario
<<Function>> Start game
<<extends>>
<<LoadScene>> Load scene 1-1
<<Function>> Quit game
<<Function>> Jump
<<includes>>
<<PlayAnimation>> Play jump animation
<<Function>> Walk
<<PlayAnimation>> Play walk animation
<<Function>> Run
<<PlayAnimation>> Play run animation
mario collides with mushroom
<<Function>> Enlarge character
<<Function>> Get power up

Fig. 3 Excerpt of a game use case model created using GUCM tool

shows an excerpt of a game use case modeled using GUCM tool, illustrating elements linkage and connection.

2. *Elements of the textual description* Each model can be described textually. For example, Fig. 4 shows the textual descriptions for *Mario* PlayerCharacter actor (Fig. 4a) and *Get Power Up* Function Game Use Case (Fig. 4b). The game use case's textual description GUI shows the extension use cases along with their associated extension points (see Fig. 4b). An extension point is characterized by its name and its triggering condition. Figure 5 illustrates the extension point textual description GUI showing the extension point name, its triggering condition/event and source/destination game use cases. In addition, game use case sketches can be added and removed through the GUI interface of the use case textual description.
3. *Managing game use case sketches* GUCM allows the modeler to upload, update, and remove game use case sketches (within the textual description of a game use case). In addition, it supports zooming and scrolling through the game use case sketch. Figure 6 shows a sketch preview in GUCM of Walk Animation Game Use Case.
4. *Flexible connectors of elements* The links between model elements can be extended through line points

Fig. 4 Example of game actor/use case textual descriptions using GUCM tool



Game Actor Textual Description

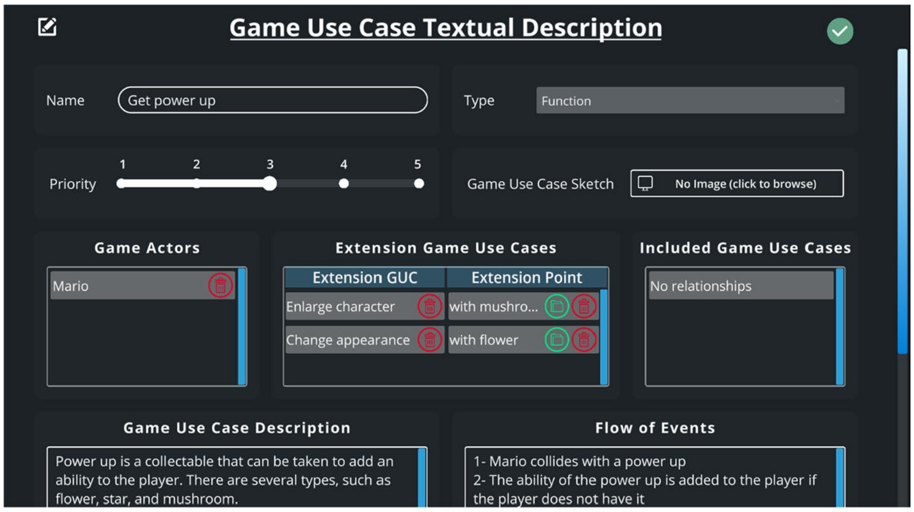
Name: Mario

Type: Player Character

Game Actor Description: Main character of the game

Game Use Cases: Jump, Walk, Run, Get power up

(a) Game Actor textual description using GUCM tool



Game Use Case Textual Description

Name: Get power up

Type: Function

Priority: 1 2 3 4 5

Game Use Case Sketch: No Image (click to browse)

Game Actors: Mario

Extension Game Use Cases

Extension GUC	Extension Point
Enlarge character	with mushro...
Change appearance	with flower

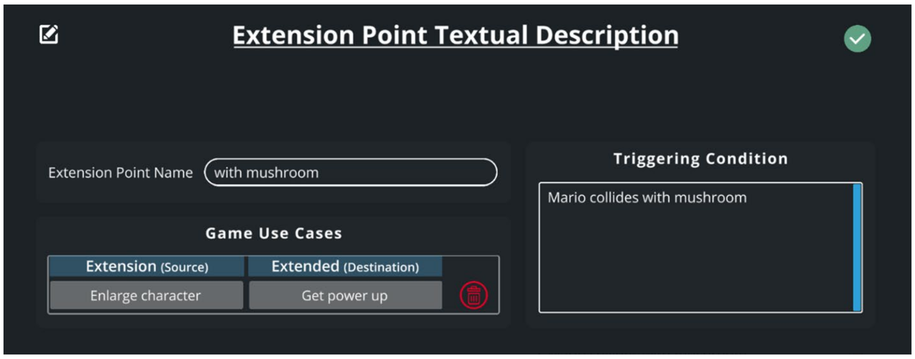
Included Game Use Cases: No relationships

Game Use Case Description: Power up is a collectable that can be taken to add an ability to the player. There are several types, such as flower, star, and mushroom.

Flow of Events: 1- Mario collides with a power up 2- The ability of the power up is added to the player if the player does not have it

(b) Game Use Case textual description using GUCM tool

Fig. 5 Example of extension point textual description using GUCM tool



Extension Point Textual Description

Extension Point Name: with mushroom

Triggering Condition: Mario collides with mushroom

Game Use Cases

Extension (Source)	Extended (Destination)
Enlarge character	Get power up

(green dots in Fig. 3), allowing for a flexible and better user experience. Line points can be moved by dragging them along the model element. They can also be deleted.

5. *Error notification system* GUCM notifies the user when a strict rule is being violated and prevents him from violating that rule by rejecting the user's action. Figure 7 shows an example of an error message triggered when the user tries to connect an actor to the same use case twice. In addition, GUCM notifies the user about violations of soft rules and displays a textual warning. For example, Fig. 8a shows the number of existing warnings (displayed next to the notification button (on the top right of the window)). Figure 8b depicts the warning textual descriptions and recommendations to resolve these inconsistencies. These notifications are listed once the user clicks the notification button.
6. *Game use case model management* GUCM allows users to save (☁) their models for later retrieval by loading

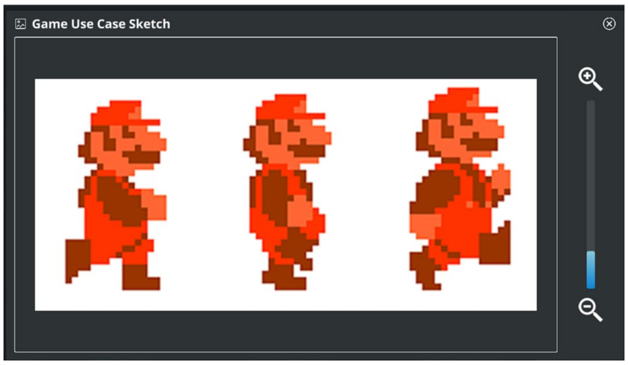


Fig. 6 Example of game use case sketch preview using GUCM tool

the models from device (☁). Models are saved using json representation (.json file) in any location the users chooses using browse directory ability.

7. *Exporting game use case models* GUCM allows users to export their models to .png file format using the screenshot button (📷).

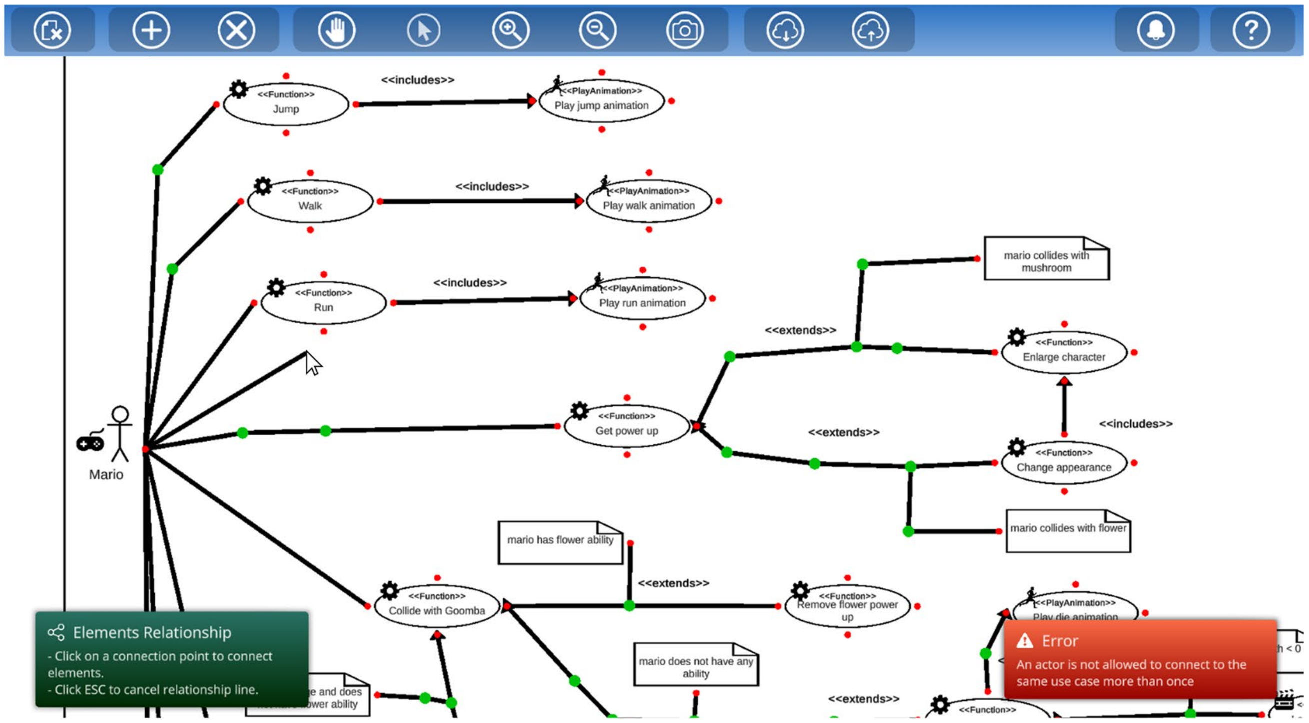
6 Applying our proposed use case-based game modeling approach

In this section, we demonstrate the applicability of our approach using four well-known games, Super Mario Bros [28], Tetris [29], Just Dance [30], and The Walking Dead [31]. We illustrate the usage of different types of actors and game use cases. However, our proposed technique is not limited to these four games and can be applied to many genres of games. It is worth noting that the presented diagrams are for illustration only and they do neither cover fully all aspects nor all components of the Super Mario Bros, Tetris, Just Dance, and The Walking Dead games.

6.1 Illustrative example 1: Super Mario Bros

Super Mario Bros is a 2D platform game (a.k.a. platformer) having *Mario* as its main character. Mario, controlled by the player, must jump and climb between suspended platforms while avoiding obstacles. Its objective is to reach the end of each level, survive the main antagonist Bowser's forces (killing the enemies), and save Princess Peach.

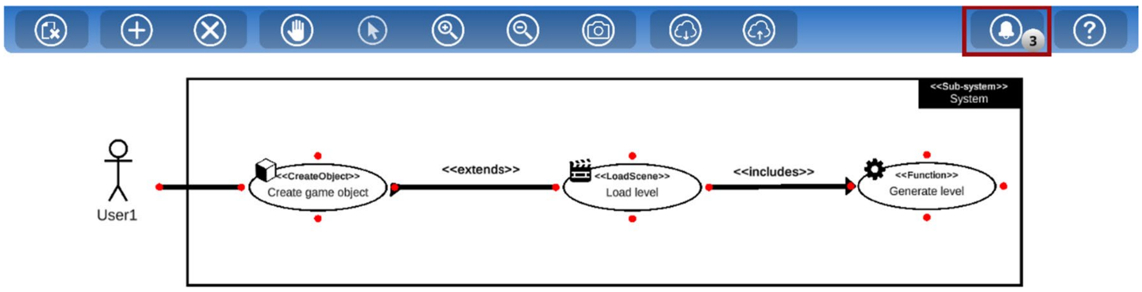
Figure 9 illustrates the game use case model of *Mario* character in Super Mario Bros game. Mario is represented as a *PlayerCharacter* actor (a joystick is shown next to the actor stick). Mario's functionalities/behaviors are expressed using eight use cases of type ≪Function≫, namely Jump,


<<Function>> Jump
<<includes>>
<<PlayAnimation>> Play jump animation
<<Function>> Walk
<<includes>>
<<PlayAnimation>> Play walk animation
<<Function>> Run
<<includes>>
<<PlayAnimation>> Play run animation
mario collides with mushroom
<<extends>>
<<Function>> Enlarge character
<<Function>> Get power up
<<extends>>
<<includes>>
<<Function>> Change appearance
mario collides with flower
Mario
mario has flower ability
<<extends>>
<<Function>> Collide with Goomba
<<Function>> Remove flower power up
<<PlayAnimation>> Play die animation
mario does not have any ability
<<extends>>
<<Function>>

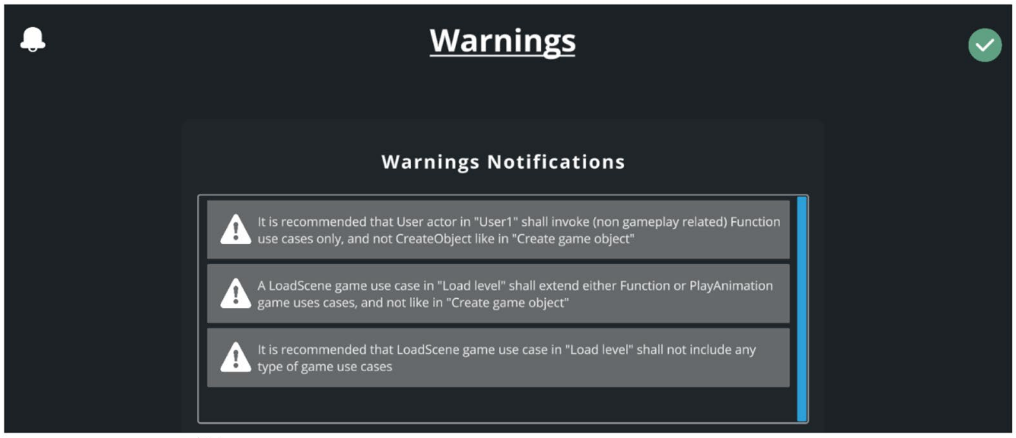
Elements Relationship
- Click on a connection point to connect elements.
- Click ESC to cancel relationship line.

Error
An actor is not allowed to connect to the same use case more than once

Fig. 7 An example of an error message in GUCM tool



(a) A game use case model showing three warnings



Warnings

Warnings Notifications

It is recommended that User actor in "User1" shall invoke (non gameplay related) Function use cases only, and not CreateObject like in "Create game object"

A LoadScene game use case in "Load level" shall extend either Function or PlayAnimation game uses cases, and not like in "Create game object"

It is recommended that LoadScene game use case in "Load level" shall not include any type of game use cases

(b) Example of warnings descriptions and recommendations

Fig. 8 An example of a model violating three soft rules

Walk, Run, Throw fire ball, Get power up, Go into pipe, Win, and Collide with Goomba.

Each *Function* use case either includes or is extended by other game use cases supplied by different development teams, e.g., animation team. For example, the "Jump" use case (of type ≪Function≫) is used to describe how jumping, falling, and landing (the three main states of a jump) occur. The jump should be coordinated with jump animation described using use case "Play jump animation" (of type ≪PlayAnimation≫). This use case is included (using ≪includes≫ relationship) within the "Jump" use case. Figure 10 shows the textual descriptions of "Jump," while Fig. 11 illustrates the "Jump Animation" game use case of Mario character as presented in Fig. 9. Furthermore, "change appearance" (of type ≪Function≫) extends "Get power up" function. It is executed when the event (expressed as an extension point) "mario collides with flower" takes place.

One of the important concepts to understand when developing a game use case model is the ≪CreateObject≫ use case type. To use this type, we assume that the definition of the object to be created is already available. Thus, in the model depicted in Fig. 9, we can see that *fireball* is represented by a *NonPlayerCharacter* and it has its own use cases. In addition, the *fireball* actor has undirected link with "Create fire ball" game use case (of type ≪CreateObject≫), which creates an instance of the fireball and which is included by "Throw fire ball" function game use case that is invoked by Mario *PlayerCharacter*.

Furthermore, Mario character triggers a use case of type ≪LoadScene≫ in the following four situations: (1) Mario is winning the current level and moving to the next one, i.e., "Load next level," (2) Mario is going into a pipe which loads a secret scene, "Load secret scene," (3) Mario dies and no more health score is left, causing the main menu scene to load, i.e., "Load main scene," and (4) Mario dies and his health score is greater than the lower boundary which reloads the current level, i.e., "Load current level." Figure 12 shows the textual description of "Die" Function game use case along with their corresponding use case extensions, i.e., "Load current level" and "Load main scene." Figure 13 shows the details of "has health" extension point.

Furthermore, Fig. 9 describes the use cases associated with Goomba mushroom enemy character in Super Mario Bros. Goomba character is represented as a "NonPlayerCharacter" actor and has two main ≪Function≫ game use cases, namely "Walk goomba" and "Destroy goomba." Each one of them includes a ≪PlayAnimation≫ game use case. In addition, the "Destroy goomba" Function game use case

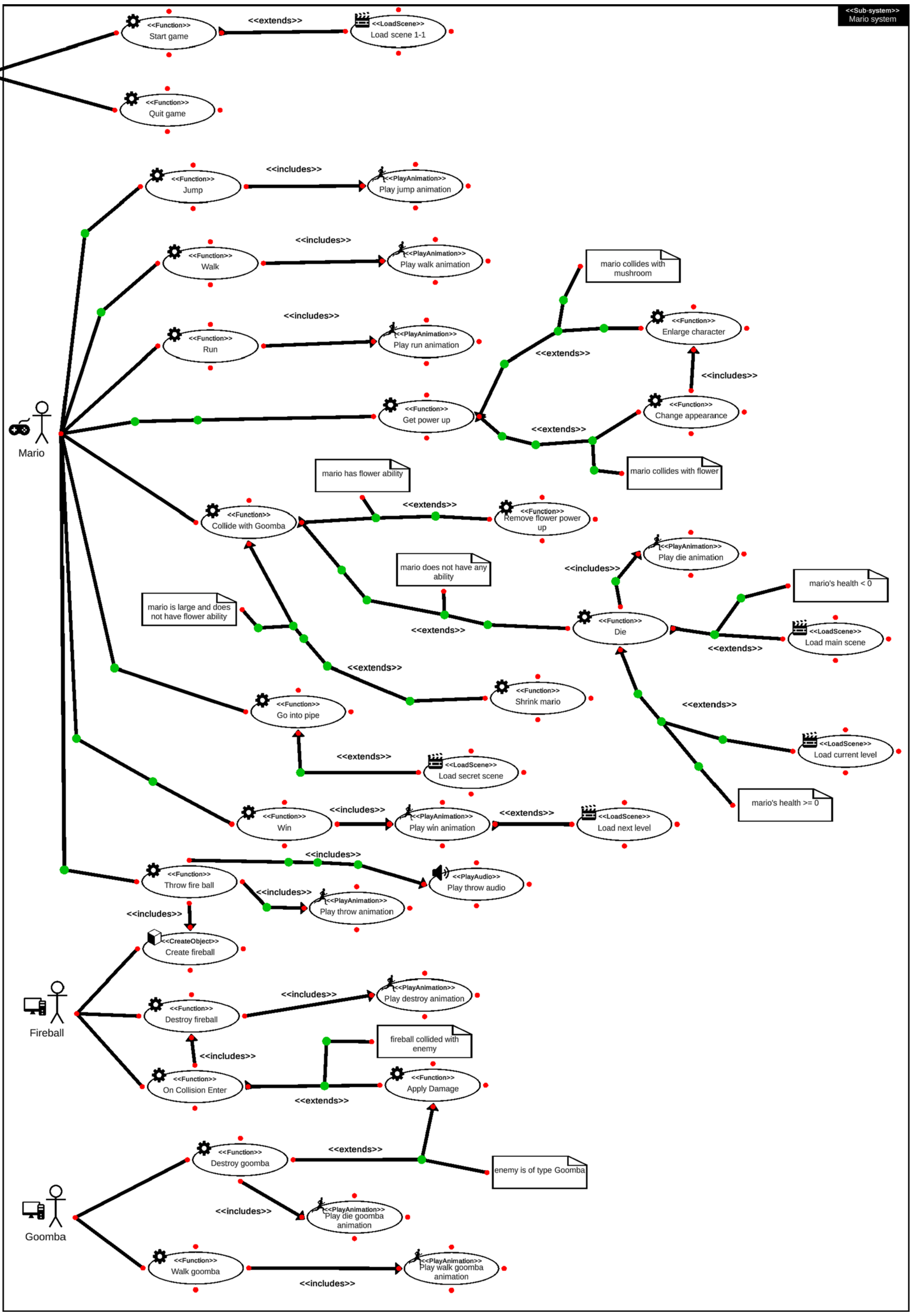


Fig. 9 Game Use Case Model of Super Mario Bros produced using GUCM tool

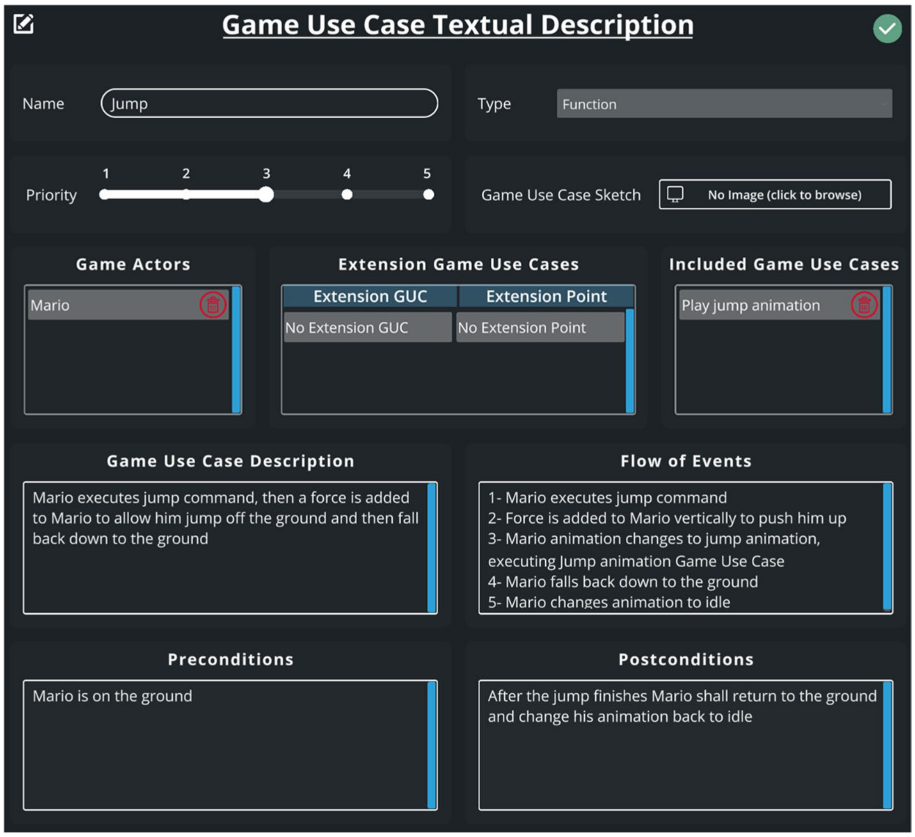


Fig. 10 Use case textual description of Mario's "Jump" game use case

Game Use Case Textual Description

Name	Jump
Type	Function
Priority	3
Game Use Case Sketch	No Image (click to browse)

Game Actors

Mario

Extension Game Use Cases

Extension GUC	Extension Point
No Extension GUC	No Extension Point

Included Game Use Cases

Play jump animation

Game Use Case Description

Mario executes jump command, then a force is added to Mario to allow him jump off the ground and then fall back down to the ground

Flow of Events

1- Mario executes jump command
2- Force is added to Mario vertically to push him up
3- Mario animation changes to jump animation, executing Jump animation Game Use Case
4- Mario falls back down to the ground
5- Mario changes animation to idle

Preconditions

Mario is on the ground

Postconditions

After the jump finishes Mario shall return to the ground and change his animation back to idle

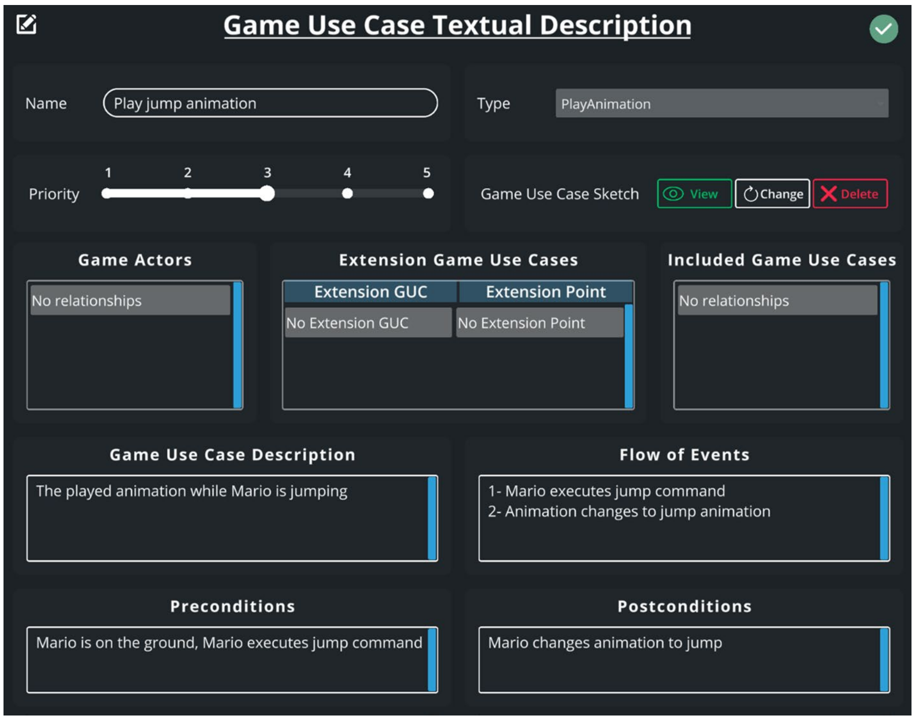


Fig. 11 Use Case textual description of "Play Jump animation" game use case

Game Use Case Textual Description

Name	Play jump animation
Type	PlayAnimation
Priority	3
Game Use Case Sketch	View / Change / Delete

Game Actors

No relationships

Extension Game Use Cases

Extension GUC	Extension Point
No Extension GUC	No Extension Point

Included Game Use Cases

No relationships

Game Use Case Description

The played animation while Mario is jumping

Flow of Events

1- Mario executes jump command
2- Animation changes to jump animation

Preconditions

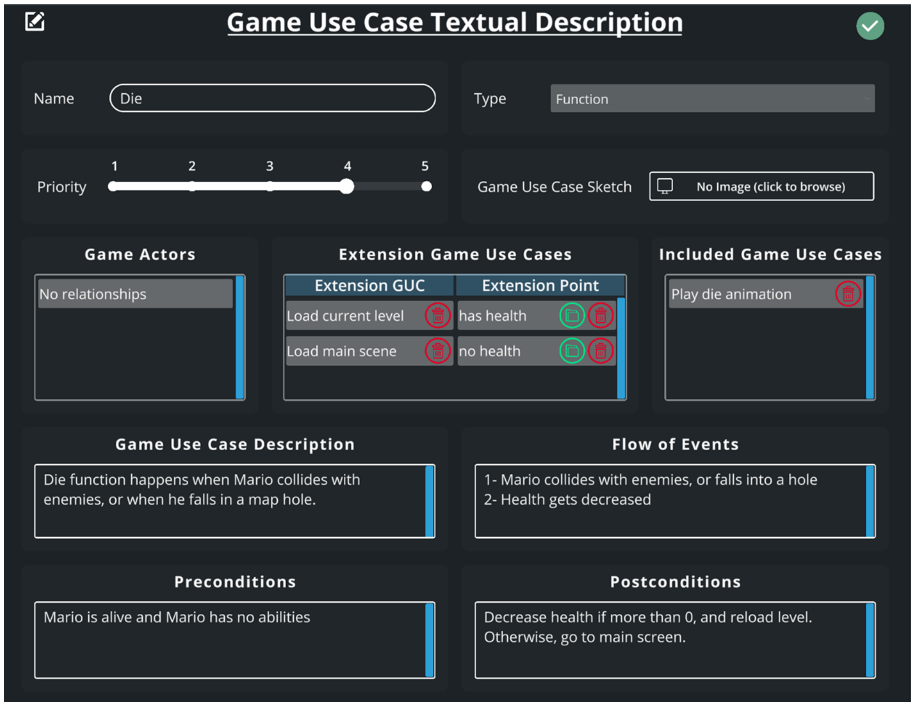
Mario is on the ground, Mario executes jump command

Postconditions

Mario changes animation to jump

extends "Apply Damage" game use case which is triggered if fireball collides with an enemy of type goomba.

Figure 9 illustrates an actor of type "User" along with two ≪Function≫ game use cases, i.e., "Start game" and "Quit



Game Use Case Textual Description

Name: Die

Type: Function

Priority: 1 2 3 4 5 (4)

Game Use Case Sketch: No Image (click to browse)

Game Actors

No relationships

Extension Game Use Cases

Extension GUC	Extension Point
Load current level	has health
Load main scene	no health

Included Game Use Cases

Play die animation

Game Use Case Description

Die function happens when Mario collides with enemies, or when he falls in a map hole.

Flow of Events

1- Mario collides with enemies, or falls into a hole
2- Health gets decreased

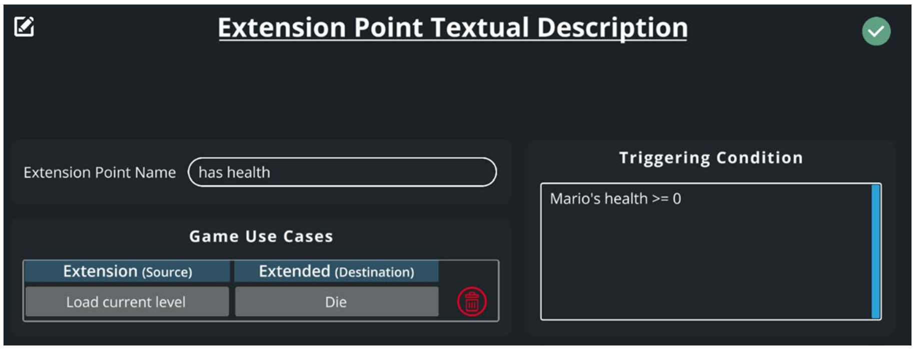
Preconditions

Mario is alive and Mario has no abilities

Postconditions

Decrease health if more than 0, and reload level. Otherwise, go to main screen.

Fig. 12 Use case textual description of Mario's "Die" game use case



Extension Point Textual Description

Extension Point Name: has health

Triggering Condition

Mario's health >= 0

Game Use Cases

Extension (Source)	Extended (Destination)
Load current level	Die

Fig. 13 Extension point textual description "has health"

game." The "Start game" use case is extended by a ≪LoadScene≫ game use case that loads the first level of the game.

The model represented in this example does not only cover aspects of game's functions, but it also captures game play and game design aspects, where the model shows PlayerCharacter's abilities, rules of winning, rules of loosing, rules of getting power ups, and rules of taking damage. In addition, the model shows some perspectives related to game's secrets. Moreover, more details can be added to the model by utilizing the game use case textual description GUI.

6.2 Illustrative example 2: Tetris

Tetris [29] is an arcade puzzle game. The goal of the game is to match tiles and create lines to achieve high scores and win levels. Figure 14 shows a game use case model representation of Tetris. A Tetris piece is modeled as a "PlayerCharacter" actor, where a piece is a game object controlled by the player.

The tetris piece can perform two use cases of type ≪ Function≫: (1) *Move* (for moving the piece in the game board, and (2) *Rotate* (to rotate the piece). The *Place* game use case of type ≪Function≫ extends *Move* and places the piece in the board. *Place* use case includes *Stop control* use case of type ≪Function≫ and *Play placement audio* use case of type ≪PlayAudio≫. The game ends when the use case *Lose* of type ≪Function≫ is executed, which is triggered once the placed piece's position exceeds the upper board bar.

The player wins if the score is greater than or equal to the level's score. If the placed piece did not exceed the upper bar and the state is not a wining state, then a new Tetris piece is created with the *Create next piece* use case of type ≪ CreateObject≫, which takes control from the lastly placed

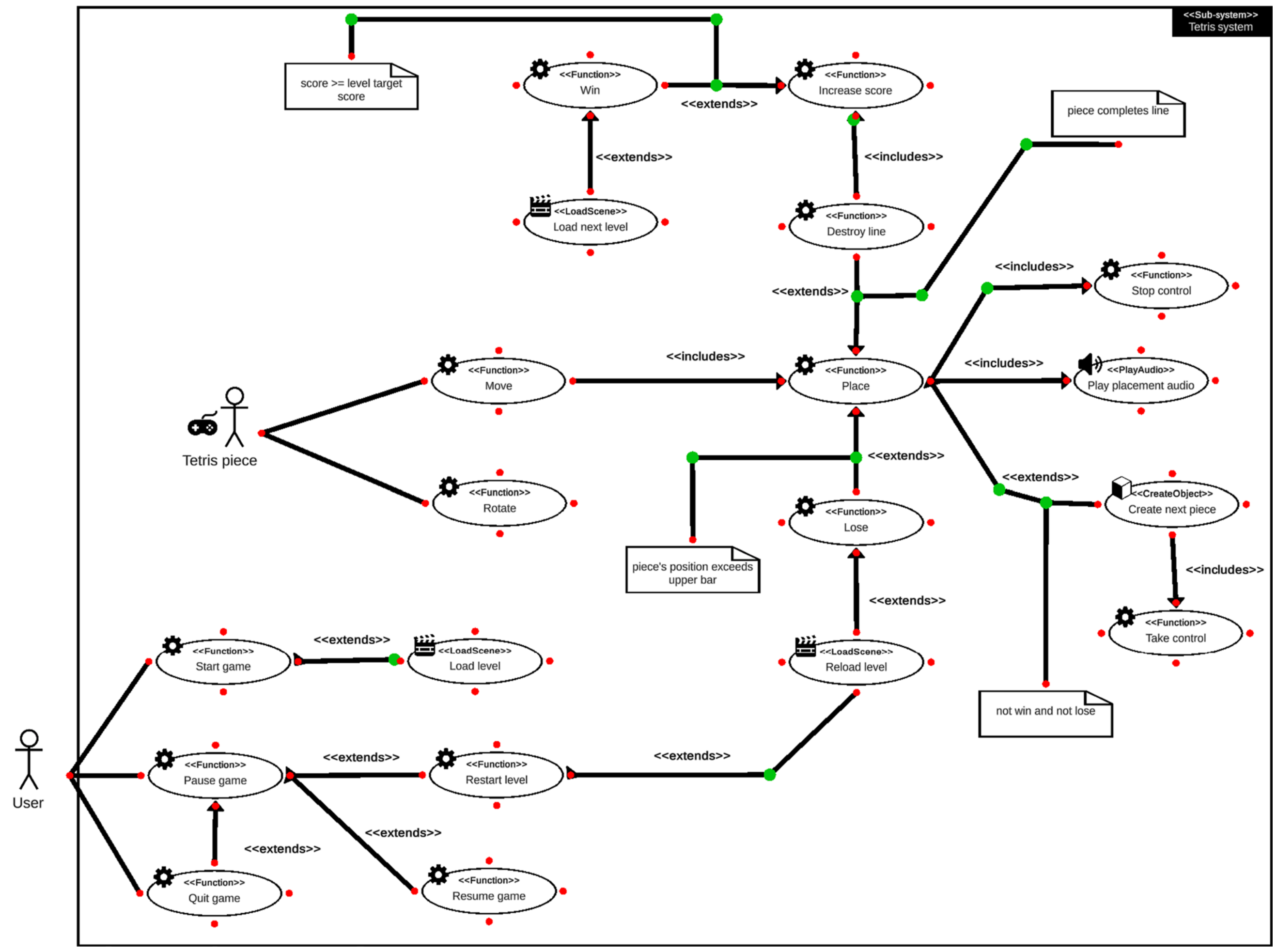

<<Sub-system>> Tetris system
score >= level target score
<<Function>> Win
<<Function>> Increase score
<<extends>>
<<LoadScene>> Load next level
<<Function>> Destroy line
<<includes>>
piece completes line
<<Function>> Stop control
<<PlayAudio>> Play placement audio
<<Function>> Place
<<Function>> Move
<<Function>> Rotate
Tetris piece
<<Function>> Lose
piece's position exceeds upper bar
<<CreateObject>> Create next piece
<<Function>> Take control
<<LoadScene>> Reload level
not win and not lose
<<Function>> Start game
<<LoadScene>> Load level
<<Function>> Pause game
<<Function>> Restart level
<<Function>> Quit game
<<Function>> Resume game
User

Fig. 14 Game use case model of Tetris game

piece. Figure 15 depicts the textual description for Rotate game use case.

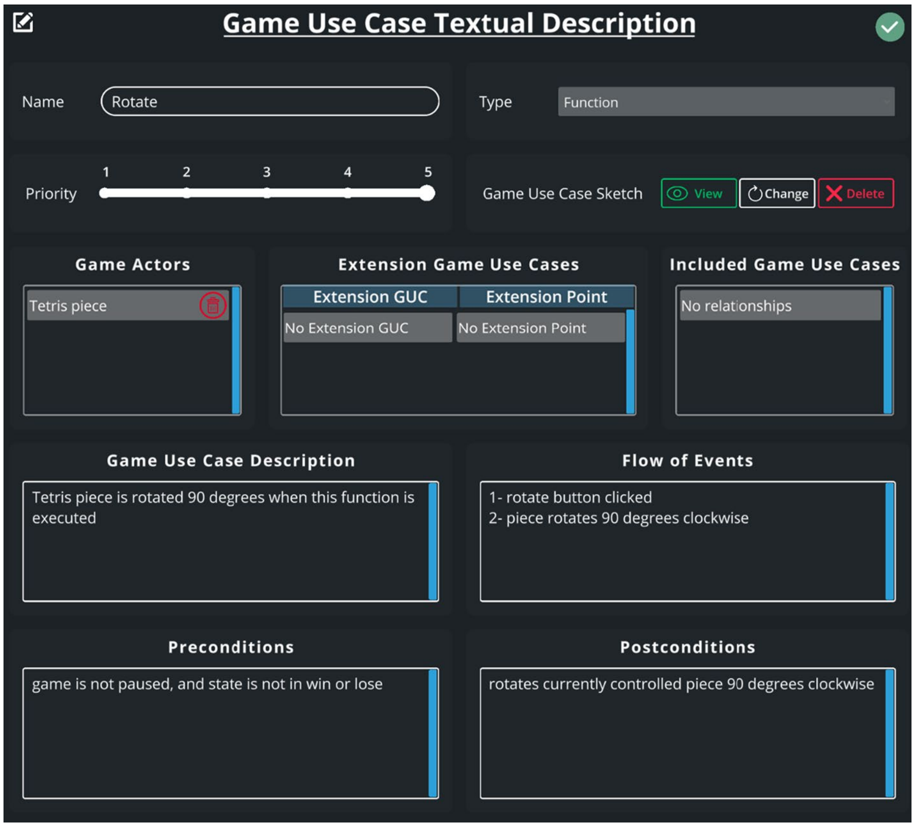
6.3 Illustrative example 3: just dance

Just Dance [30] is a music, rhythm, and dance game, where player's follow dance moves based on a playing song, and they compete based on scores given to the most matching dancer. Figure 16 shows a conceptual game use case model representation of Just Dance that represents the game play aspects of the game. In this game, a "PlayerCharacter" actor that represents the tracked player was modeled to be controlling the game by physically copying dance moves through applying *Copy dance move* ≪Function≫ game use case. The dance moves to be copied are provided by "NonPlayerCharacter" actor (Just dance system), which plays animation of dance moves that extend playing the music. Moreover, it evaluates the moves performed by the player through *Detect player's dance move* ≪Function≫ game use case that in-turn includes *Calculate dance move score* ≪Function≫ game use case. When the music stops, the "NonPlayerCharacter" actor (Just dance system) evaluates and rates player's overall dance moves through two ≪Function≫ game use cases, *Rate player's performance* and *Calculate player's score*.

Game play aspects can be modeled using our game use case modeling approach through combining different types of game use cases as shown in Just Dance and Mario examples. In addition, the modeler can utilize extension points to express game rules and can also use the game use case textual description to elaborate more on the game play using preconditions, post-conditions, and flow of events.

6.4 Illustrative example 4: the walking dead

The Walking Dead [31] is an episodic adventure interactive drama game that has a flavor of role-playing (RPG), where the player can take choices that could affect the flow of story and events. The game consists of several scenes and a variety of characters. However, in this example, and similar to the other illustrative examples, we have just modeled some



Name	Rotate
Type	Function
Priority	1 2 3 4 5
Game Use Case Sketch	

Game Actors

Tetris piece

Extension Game Use Cases

Extension GUC	Extension Point
No Extension GUC	No Extension Point

Included Game Use Cases

No relationships

Game Use Case Description

Tetris piece is rotated 90 degrees when this function is executed

Flow of Events

1- rotate button clicked
2- piece rotates 90 degrees clockwise

Preconditions

game is not paused, and state is not in win or lose

Postconditions

rotates currently controlled piece 90 degrees clockwise

Fig. 15 Textual description of Tetris Rotate Function use case

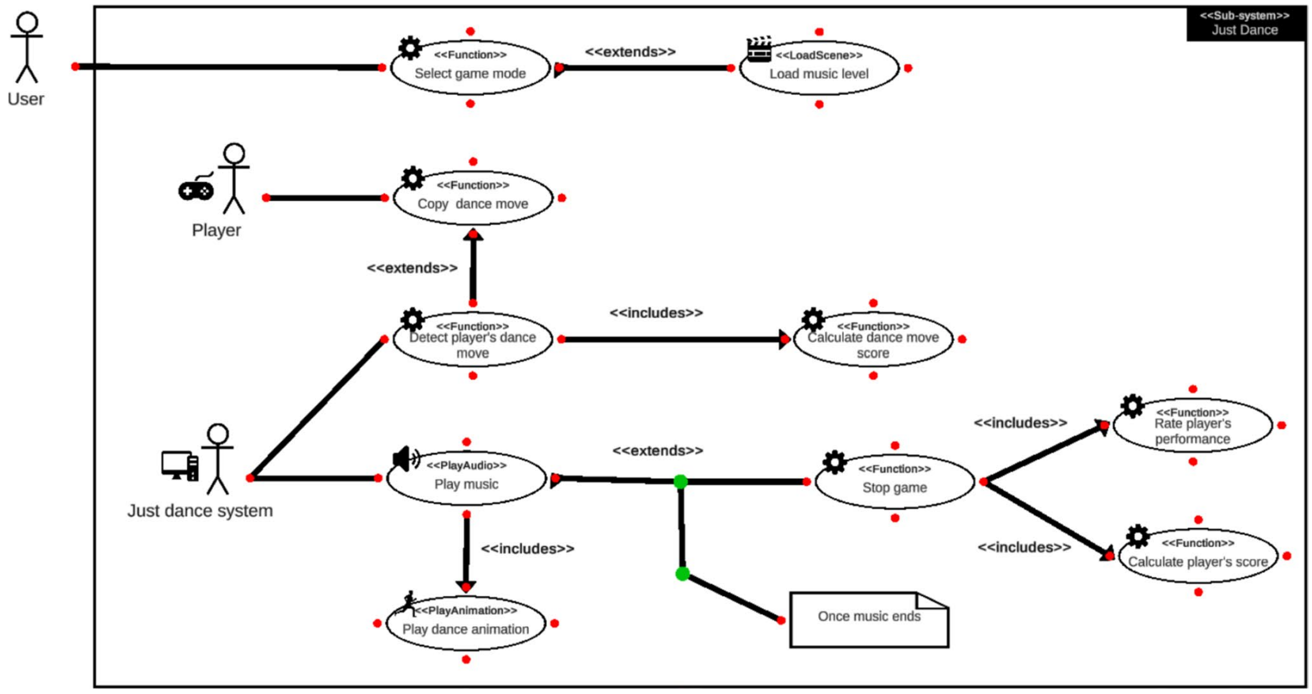


Fig. 16 Game use case model of Just Dance game

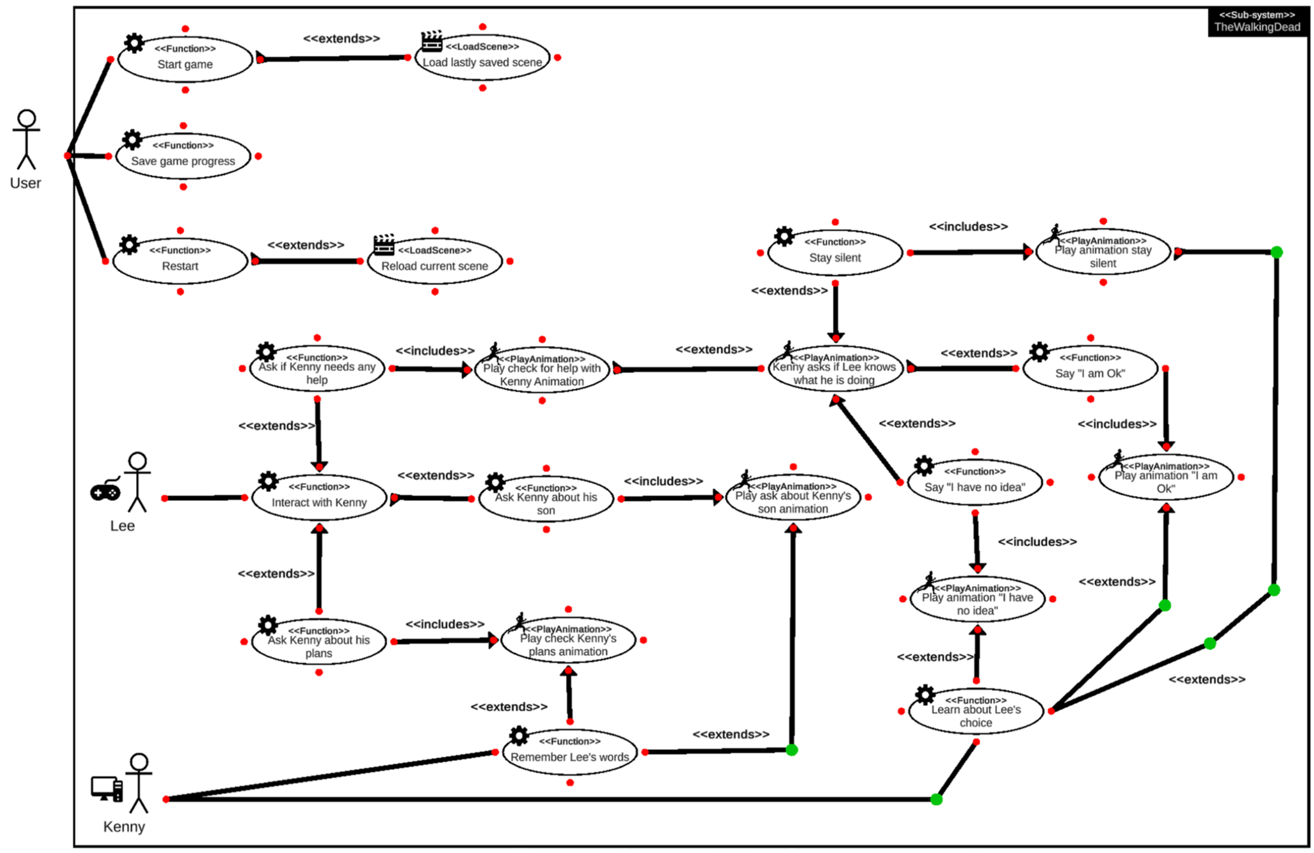

<<Sub-system>>
TheWalkingDead
User
<<Function>> Start game
<<extends>>
<<LoadScene>> Load lastly saved scene
<<Function>> Save game progress
<<Function>> Restart
<<extends>>
<<LoadScene>> Reload current scene
<<Function>> Stay silent
<<includes>>
<<PlayAnimation>> Play animation stay silent
<<extends>>
<<Function>> Ask if Kenny needs any help
<<includes>>
<<PlayAnimation>> Play check for help with Kenny Animation
<<extends>>
<<PlayAnimation>> Kenny asks if Lee knows what he is doing
<<extends>>
<<Function>> Say "I am Ok"
<<extends>>
<<extends>>
<<includes>>
Lee
<<Function>> Interact with Kenny
<<extends>>
<<Function>> Ask Kenny about his son
<<includes>>
<<PlayAnimation>> Play ask about Kenny's son animation
<<Function>> Say "I have no idea"
<<PlayAnimation>> Play animation "I am Ok"
<<includes>>
<<PlayAnimation>> Play animation "I have no idea"
<<extends>>
<<extends>>
<<Function>> Ask Kenny about his plans
<<includes>>
<<PlayAnimation>> Play check Kenny's plans animation
<<extends>>
<<extends>>
<<Function>> Learn about Lee's choice
<<extends>>
<<Function>> Remember Lee's words
<<extends>>
Kenny

Fig. 17 Game use case model of The Walking Dead game

aspects of the game. In this example, we model players' interactions, dialogues, and choice selection.

Figure 17 illustrates the interaction between a "Player-Character" actor *Lee* and a "NonPlayerCharacter" actor *Kenny*, where a dialogue and choices of actions/replies could be taken by the player which might affect the flow of the game. For example, when the player interacts with Kenny, one of three choices can be taken by the player, either to *Ask if Kenny needs any help*, *Ask Kenny about his son*, or *Ask Kenny about his plans* ≪Function≫ game use cases.

It is worth noting that our modeling approach does not model or represent menu items. However, it models the functions behind the menu items; thus, in this example each choice is modeled as a ≪Function≫ game use case. If the modeler needs to add details related to how the dialogue and choices appear in the game, a sketch of that game use case can be added through its textual description.

As shown in Fig. 17, after a selection is made, a sequence of game use cases take place. For example, if the player chooses *Ask if Kenny needs any help* ≪Function≫ game use case, a sequence of animations is played, and is followed by another set of choices that lead to remembering the conversation by the "NonPlayerCharacter" actor *Kenny* through *Learn about Lee's choice* ≪Function≫ game use case.

It is worth noting that some use cases are linked to more than one use case, e.g., Play animation "I am Ok" game use case is linked, through an include relationship, to Say "I am Ok" and through an extend relationship to Learn about Lee's choice. Managing complexity is an inherent problem in use case modeling. Modelers have to carefully decompose complex behavior (using UC relationships) taking into account all possible execution scenarios (e.g., covering all possible values of the extension points' conditions). In addition, a special care is needed in order to maintain an acceptable level of granularity and minimize the impact of future changes. Indeed, in presence of many interconnected use cases, a change in one use case may require changes to the linked use cases.

7 Empirical validation

In this section, we validate empirically our proposed use case-based game modeling approach. In particular, we aim to assess the perceived learnability, understandability, and usefulness of our proposed modeling approach using an online questionnaire-based survey [66]. We report our empirical validation using the templates and recommendations presented in Wohlin et al. [67].

7.1 Experiment goals

Our proposed use case-based modeling approach enrich the standard UML use case diagram notation with many game-related constructs, in order to support the requirements modeling of games. The main goal of the conducted questionnaire-based survey is to evaluate the perceived learnability, understandability, and usefulness of our proposed modeling approach within the game development community.

The ISO 9126 [68] standard defines *learnability* and *understandability* as two sub-characteristics of *usability*. Learnability emphasizes that "the system should be easy to learn by the class of users for whom it is intended" [69]. Understandability is defined as the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use [68]. Usefulness is defined as "the degree to which a person believes that using a particular system would enhance his or her job performance" [70].

7.2 Experimental design

Figure 18 illustrates the main steps of our experimental plan.

7.2.1 Subjects

Since our use case-based modeling approach focuses mainly on enhancing requirements modeling for game development project and supporting the communication between team members, our survey aims to target participants from the game development industry. Game development professionals tend to have a better and deeper understanding of the development life cycle of video game projects, thus enabling them to provide better insights and analysis of the proposed approach.

Twitter is very popular within the gaming communities. In 2019, more than 1.2 billion tweets on the Twitter platform were related to video games [71]. The game development community on Twitter is composed of a wide variety of experienced game developers, artists, designers, etc., and is known for its active social communication. Hence, we have targeted participants from this active community to take part in the evaluation of our proposed game-oriented use case modeling approach. To ensure a good response rate, the first author of this paper tweeted and utilized active community hashtags on Twitter (#gamedev and #indiedev). In addition, we have invited participants and developers from the community to share and retweet the survey with their friends. The survey was available online for three weeks (from 8 to 27, April 2019), and we have succeeded to collect responses from a total of 29 participants.

7.2.2 Material

The material, provided online using Google Forms, consists of two parts (see Fig. 18):

1. *Study intent and introduction to the game-oriented use case notation* This part starts with a paragraph that states briefly the goal of the study, followed by a brief introduction of the main constructs (i.e., actors, use cases, relationships) of the game-oriented use case notation. Both graphical (i.e., symbols of actors and use cases) and textual (i.e., elements of the textual templates, e.g., description, priority, precondition, post-conditions, etc.) constructs were presented. The introductory material is very similar to the descriptions of game actors provided in Sect. 4.1.1 and in Table 1, and of game use cases provided in Sect. 4.1.2 and in Table 2. In addition, we have provided the participants with the use case diagram and the textual description of the Super Mario Bros [28] example, as a concrete example of the application of our approach. It is worth noting that the model given to the participants is slightly different from the one presented in Fig. 9 and that no additional descriptions of the Super Mario UC model were provided to the participants, since Super Mario Bros [28] is a well-known game within the gaming community. Participants did not use our prototype tool.
2. *Questionnaire-based survey* By following the guidelines provided by Kitchenham and Pfleeger [66], an online survey was created using Google Forms. The survey is composed 15 questions (summarized in Table 3), which are divided into five categories as follows:
    - *Respondents' characterization* Five out of the 15 questions (from question CQ1 to CQ5) are used to characterize the respondents in terms of (1) their role in the game development life cycle, e.g., game programmer, artist, game level designer, game producer, etc. (CQ1), (2) their number of years of experience (CQ2), (3) whether they have previous experience with models/diagrams in developing games (CQ3), (4) whether they have previous experience with UML modeling (CQ4), and (5) whether they have faced issues in communicating game requirements to project teammates (CQ5). CQ5 is measured using a 5 point Likert scale.
    - *Perceived understandability* Three questions (UndQ1, UndQ2, UndQ3) were devoted to assess the perceived understandability of both the graphical game-oriented use case diagram and its textual

Table 3 Survey questions

Question ID	Part 1: Respondents' characterization
CQ1	Which role describes you the best? Note: the respondent was asked to choose from the following options: game programmer, artist, game level designer, game play designer, game producer, other (to be specified by the respondent)
CQ2	How many years of experience do you have in the field of game development/design?
CQ3	Have you used diagrams/models in game development/design?
CQ4	Have you ever used UML use case modeling in game development/design?
CQ5	I have faced problems to communicate game requirements to teammates, e.g., misunderstanding, vagueness, conflict, etc. To what extent do you agree with this statement?
	Part 2: Perceived Understandability
UndQ1	The use case diagram is easy to understand. To what extent do you agree with this statement?
UndQ2	The graphical elements of the game use case diagram are self-descriptive. To what extent do you agree with this statement?
UndQ3	The game textual description helps improve the understandability of the game use case diagram. To what extent do you agree with this statement?
	Part 3: Perceived Learnability
LQ1	I believe that I can learn this technique quickly. To what extent do you agree with this statement?
	Part 4: Perceived Usefulness
UQ1	The diagram would help the team to become more effective. To what extent do you agree with this statement?
UQ2	The game use case textual description is informative. To what extent do you agree with this statement?
UQ3	In your opinion, which team role(s) would benefit the most from this technique? Note: the respondent was asked to choose from the following options: game programmer, artist, game level designer, game play designer, game producer, other (to be specified by the respondent)
UQ4	The technique would improve the communication of game requirements between team members. To what extent do you agree with this statement?
UQ5	The technique would help the team to become more productive. To what extent do you agree with this statement?
	Part 5: Potential improvements
IQ1	Please provide your suggestions to improve the proposed technique?

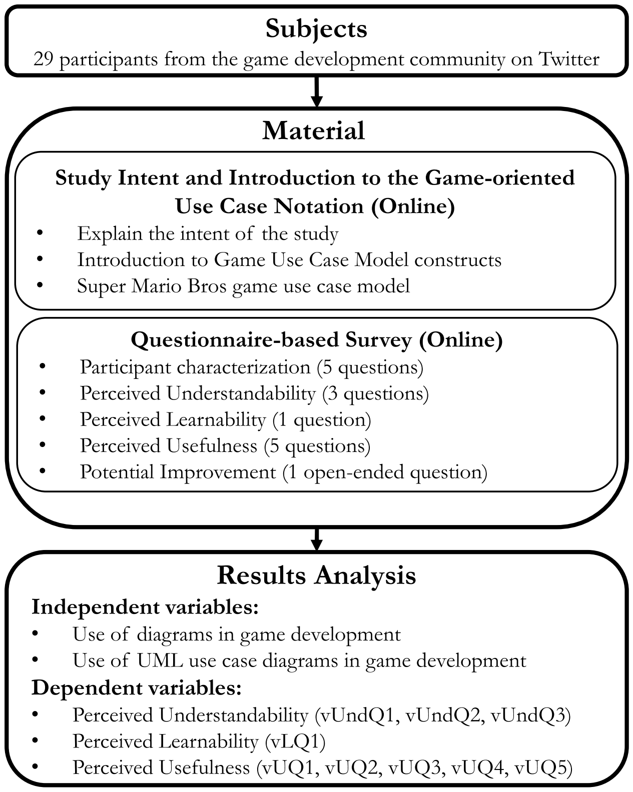

Subjects
29 participants from the game development community on Twitter
Material
Study Intent and Introduction to the Game-oriented Use Case Notation (Online)
Explain the intent of the study
Introduction to Game Use Case Model constructs
Super Mario Bros game use case model
Questionnaire-based Survey (Online)
Participant characterization (5 questions)
Perceived Understandability (3 questions)
Perceived Learnability (1 question)
Perceived Usefulness (5 questions)
Potential Improvement (1 open-ended question)
Results Analysis
Independent variables:
Use of diagrams in game development
Use of UML use case diagrams in game development
Dependent variables:
Perceived Understandability (vUndQ1, vUndQ2, vUndQ3)
Perceived Learnability (vLQ1)
Perceived Usefulness (vUQ1, vUQ2, vUQ3, vUQ4, vUQ5)

Fig. 18 Experimental design

representation. These questions are closed-ended and measured using a 5 point Likert scale.

- *Perceived learnability* One question (LQ1) is devoted to assess the perceived learnability of the technique. This question is closed-ended and measured using a 5-point Likert scale.
- *Perceived usefulness* Five questions (UQ1, UQ2, UQ3, UQ4, UQ5) were formulated in order to capture how participants perceive usefulness of the proposed technique. These questions are closed-ended and measured using a 5 point Likert scale.
- *Potential improvements* Finally, we added an open-ended question (IQ1) to collect potential improvements of the proposed technique.

7.2.3 Variables

The independent variables for our experiment are: (1) the familiarity of the respondent with diagrams/models in game development, (2) the familiarity of the respondent with use case modeling.

We assess the perceived learnability, understandability, and usefulness by the means of the dependent variables representing the opinions of the respondents to the corresponding questions.

7.2.4 Hypotheses

In addition to the analysis of the descriptive statistics of all dependent variables, the experiment was planned with the purpose of testing the following hypothesis:

1. *Perceived understandability*
   - *Perceived Understandability Hypothesis 1*
     - *H0-perceived-understandability-1* (Null hypothesis): There is no difference in the perceived understandability between participants having previous experience with diagrams/models in game development/design and participants who haven't.
     - *H1-perceived-understandability-1* (Alternative hypothesis): There is a significant difference in the perceived understandability between participants having previous experience with diagrams/models in game development/design and participants who haven't.
     - *Independent variables*: Boolean variable capturing whether the participant has used diagrams/models in game development (collected from responses to question CQ3).
     - *Dependent variables* Three variables: (1) **vUndQ1**: measures the perceived understandability of use case diagram (collected from responses to question UndQ1), (2) **vUndQ2**: measures the perception of the self-descriptiveness of the diagram graphical elements (collected from responses to question UndQ2), and (3) **vUndQ3**: measures the perceived improvement brought by the textual description (collected from responses to question UndQ3).
   - *Perceived understandability Hypothesis 2*
     - *H0-perceived-understandability-2* (Null hypothesis) There is no difference in the perceived understandability between participants having previous experience with UML use case modeling in game development/design and participants who have used diagrams other than UML use case diagrams.
     - *H1-perceived-understandability-2* (Alternative hypothesis) There is a significant difference in the perceived understandability between participants having previous experience with UML use case modeling in game development/design and participants who have used diagrams other than UML use case diagrams.
     - *Independent variables* Boolean variable capturing whether the participant has used UML use cases models or used other types of diagrams. It is retrieved from the intersection of responses to questions CQ3 and CQ4.
     - *Dependent variables* Three variables: (1) **vUndQ1**: measures the perceived understandability of use case diagram (collected from responses to question UndQ1), (2) **vUndQ2**: measures the perception of the self-descriptiveness of the diagram graphical elements (collected from responses to question UndQ2), and (3) **vUndQ3**: measures the perceived improvement brought by the textual description (collected from responses to question UndQ3).
2. *Perceived learnability*
   - *Perceived learnability Hypothesis 1*
     - *H0-perceived-learnability-1* (Null hypothesis) There is no difference in the perceived learnability between participants having previous experience with diagrams/models in game development/design and participants who haven't.
     - *H1-perceived-learnability-1* (Alternative hypothesis) There is a significant difference in the perceived learnability between participants having previous experience with diagrams/models in game development/design and participants who haven't.
     - *Independent variables* Boolean variable capturing whether the participant has used diagrams/models in game development (collected from responses to question CQ3).
     - *Dependent variables* Variable **vLQ1** used to measure the perceived learnability of the technique (collected from responses to question LQ1).
   - *Perceived learnability Hypothesis 2*
     - *H0-perceived-learnability-2* (Null hypothesis) There is no difference in the perceived learnability between participants having previous experience with UML use case modeling in game development/design and participants who have used diagrams other than UML use case diagrams.
     - *H1-perceived-learnability-2* (Alternative hypothesis) There is a significant difference in the perceived learnability between participants

having previous experience with UML use case modeling in game development/design and participants who have used diagrams other than UML use case diagrams.
- *Independent variables* Boolean variable capturing whether the participant has used UML use cases models or used other types of diagrams. It is retrieved from the intersection of responses to questions CQ3 and CQ4.
- *Dependent variables* Variable **vLQ1** used to measure the perceived learnability of the technique (collected from responses to question LQ1).

3. *Perceived usefulness*

- *Perceived usefulness Hypothesis 1*

  - *H0-perceived-usefulness-1* (Null hypothesis) There is no difference in the perceived usefulness between participants having previous experience with diagrams/models in game development/design and participants who haven't.
  - *H1-perceived-usefulness-1* (Alternative hypothesis) There is a significant difference in the perceived usefulness between participants having previous experience with diagrams/models in game development/design and participants who haven't.
  - *Independent variables* Boolean variable capturing whether the participant has used diagrams/models in game development (collected from responses to question CQ3).
  - *Dependent variables* Four variables: (1) **vUQ1**: measures the perceived usefulness of the use case diagram in making the team more effective (collected from responses to question UQ1), (2) **vUQ2**: measures the perceived usefulness of the textual description in providing informative content about the game use case (collected from responses to question UQ2), (3) **vUQ4**: measures the perceived usefulness of the technique in improving the communication of game requirements between team members (collected from responses to question UQ4), and (4) **vUQ5**: measures the perceived usefulness of the technique in helping the team to become more productive (collected from responses to question UQ5).

- *Perceived usefulness Hypothesis 2*

  - *H0-perceived-usefulness-2* (Null hypothesis) There is no difference in the perceived usefulness between participants having previous experience with UML use case modeling in game development/design and participants who have used diagrams other than UML use case diagrams.
  - *H1-perceived-usefulness-2* (Alternative hypothesis) There is a significant difference in the perceived usefulness between participants having previous experience with UML use case modeling in game development/design and participants who have used diagrams other than UML use case diagrams.
  - *Independent variables* Boolean variable capturing whether the participant has used UML use cases models or used other types of diagrams. It is retrieved from the intersection of responses to questions CQ3 and CQ4.
  - *Dependent variables* Four variables: (1) **vUQ1**: measures the perceived usefulness of the use case diagram in making the team more effective (collected from responses to question UQ1), (2) **vUQ2**: measures the perceived usefulness of the textual description in providing informative content about the game use case (collected from responses to question UQ2), (3) **vUQ4**: measures the perceived usefulness of the technique in improving the communication of game requirements between team members (collected from responses to question UQ4), and (4) **vUQ5**: measures the perceived usefulness of the technique in helping the team to become more productive (collected from responses to question UQ5).

7.3 Experiment execution and data collection

The survey material was presented (online using Google Forms) to the participants in the same order defined in Sect. 7.2.2, i.e., starting with the intent of the study, then a brief introduction to the game-oriented graphical and textual use case notation, followed by the Super Mario use case model, and finally the survey questions (in the same order presented in Table 3). Participants can navigate back and forth through the material and no time limit was imposed on respondents. The data were collected from 29 participants.

7.4 Results analysis

In this section, we analyze and discuss the collected data. In addition to the analysis of the data collected for each question, we test the hypotheses presented in Sect. 7.2.4. We use the nonparametric Mann–Whitney U test [72] to compare differences in medians between two independent categorical group, e.g., (1) using diagrams/models vs. not using diagrams/models, and (2) using UML UCD vs. using diagrams other than UML UCD. We have chosen a nonparametric method since our questions are measured on an ordinal scale

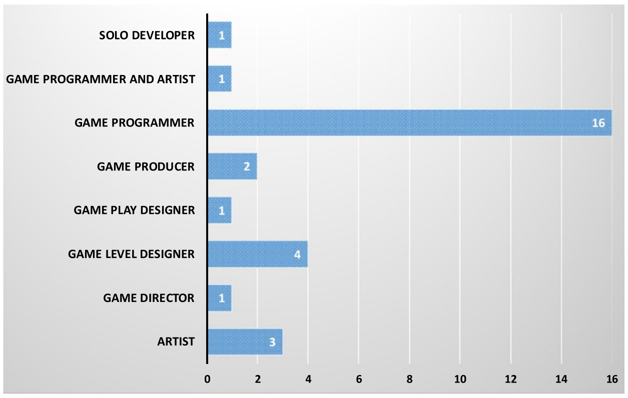


Fig. 19 Analysis of CQ1 data

(5-points Likert scale), and it is mandatory to have normally distributed data.

7.4.1 Analysis of participants' characterization questions

The first question (CQ1) is created to study the background of each participant. As shown in Fig. 19, the majority of participants were Game programmers (16 participants representing 55.2%), followed by game level designers representing 13.8%. In addition to the proposed roles, participants added three additional roles: Game programmer and artist, Game director, and Solo developer.

Assessing respondents game-related experience is addressed by question CQ2. As shown in Fig. 20, respondents with various levels of experience participated in the study, ranging from junior game developers (with less than a year of experience) to well-experienced (with more than 10 years of experience).

Question CQ3 aims to find out whether the respondents have used models and diagrams in game development and design. Results indicate that 20 respondents (representing 69% of total participants) have used diagrams/models within the game development life cycle, while 9 respondents have not.

Question CQ4 aims to find out whether the respondents have used UML use case modeling in game development and design. Results indicate that only 9 respondents (representing 31% of total participants) have used UML use case models within the game development life cycle, while 20 respondents have not. Table 8 (in "Appendix") confirms that participants who answered affirmatively question CQ4 have also answered affirmatively question CQ3 and that participants are classified into three groups: (1) no prior use of diagrams (9 respondents), (2) use of diagrams other than UML UCD (11 respondents), (3) use of UML UCD (9 respondents).

Question CQ5 aims to find out whether the respondents have faced problems to communicate game requirements to teammates (e.g., misunderstanding, vagueness, conflict, etc.). Results show that more than half of respondents (13.8% strongly agree and 41.6% agree) admitted that they have faced communication issues with respect to game requirements between team mates. Only 2 respondents disagreed (7%) while 11 respondents were neutral (38%). This supports our claim that there is an urgent need for formal (or semi-formal) techniques to describe game-related requirements.

In the following subsections, i.e., Sects. 7.4.2, 7.4.3 and 7.4.2, we present and analyze the collected data. Table 8 in "Appendix" shows the full breakdown of answers, where participants who have not used models/diagrams are coded as "0," while those who have used models/diagrams are coded as "1." Among the participants who have used diagrams/models, the ones who haven't used UML UCD are coded as "0," while those who have used UML UCD are coded as "1."

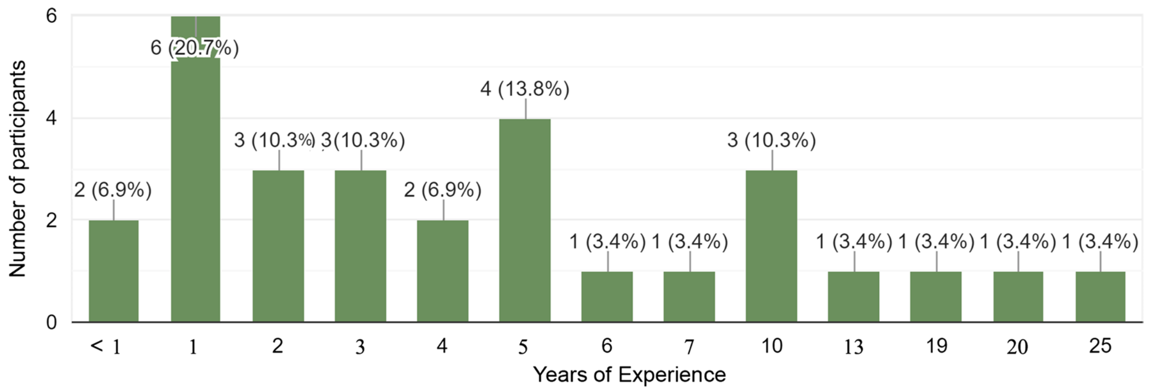


Fig. 20 Analysis of CQ2 data

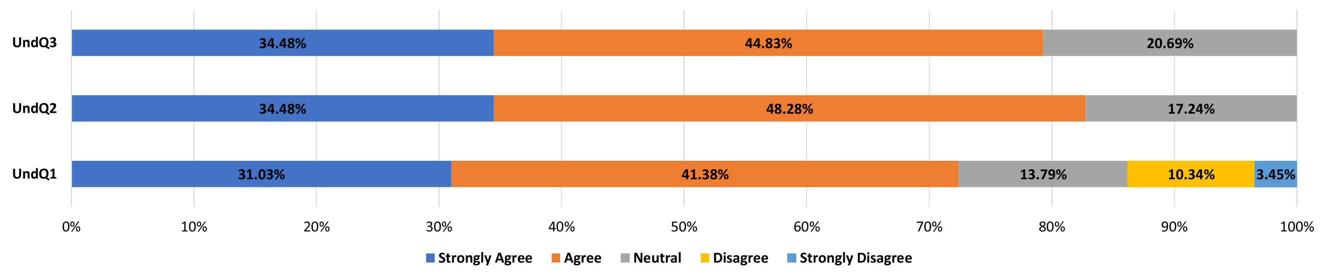


Fig. 21 Analysis of the perceived understandability data

7.4.2 Analysis of the perceived understandability data

The first question UndQ1 aims to assess whether the new game use case diagram is easy to understand in general (as perceived by the respondents). As shown in Fig. 21, 72.4% of the participants agreed (31% strongly agreed, 41.4% agreed) with the statement, while 13.7% disagreed.

Question UndQ2 aims to assess the self-descriptiveness of the game use case graphical elements (as perceived by the respondents). Results show that the majority of participants found them self-descriptive since 82.8% responded with either "agree" or "strongly agree," while only 17.2% were neutral.

The third question UndQ3 aims to assess whether the new game use case textual description is easy to understand in general (as perceived by the respondents). Results show that 79.3% found it easy to understand, while 20.7% were neutral. Hence, we can conclude that the textual description is of good support of the graphical representation.

In addition, in order to test hypotheses “Perceived Understandability 1” and “Perceived Understandability 2,” we have conducted Mann–Whitney U test [72] on responses to questions UndQ1, UndQ2, and UndQ3. Table 4 illustrates the rank table with respect to the independent variable “use of diagrams.” The group of participants with no prior use of diagrams is coded as "0," while the group of participants who have used diagram is coded as "1."

Table 5 depicts the results of Mann–Whitney U test for “Perceived Understandability 1” hypothesis. The significance values (i.e., Asym. Sig. (2-tailed)) for these three questions (i.e., 0.046, 0.04, and 0.035 are all less than $\alpha = 0.05$) show that there is a significant difference between the group of participants using diagrams/models and the group of participants not using diagrams/models, with respect to the perceived understandability. Thus, we reject the null hypothesis “H0-perceived-understandability-1” and accept the alternative hypothesis “H1-perceived-understandability-1.”

Table 6 illustrates the rank table with respect to the independent variable “use of UML UCD.” The group of participants with prior use of diagrams other than UML UCD is coded as "1," while the group of participants who have used UML UCD is coded as "2."

Table 4 Mann–Whitney U test: Rank table with respect to the independent variable “use of diagrams”

Use diagrams	N	Mean rank	Sum of ranks
UndQ1			
0	9	19.44	175.00
1	20	13.00	260.00
Total	29		
UndQ2			
0	9	19.44	175.00
1	20	13.00	260.00
Total	29		
UndQ3			
0	9	19.61	176.50
1	20	12.92	258.50
Total	29		
LQ1			
0	9	19.22	173.00
1	20	13.10	262.00
Total	29		
UQ1			
0	9	19.83	178.50
1	20	12.82	256.50
Total	29		
UQ2			
0	9	21.39	192.50
1	20	12.12	242.50
Total	29		
UQ4			
0	9	19.39	174.50
1	20	13.02	260.50
Total	29		
UQ5			
0	9	21.11	190.00
1	20	12.25	245.00
Total	29		

Table 5 Mann–Whitney U test results with respect to the independent variable "use of diagrams"

	UndQ1	UndQ2	UndQ3	LQ1	UQ1	UQ2	UQ4	UQ5
Mann–Whitney U	50.000	50.000	48.500	52.000	46.500	32.500	50.500	35.000
Wilcoxon W	260.000	260.000	258.500	262.000	256.500	242.500	260.500	245.000
Z	− 1.991	− 2.054	− 2.108	− 1.963	− 2.181	− 2.934	− 2.010	− 2.829
Asymp. Sig. (2-tailed)	.046	.040	.035	.050	.029	.003	.044	.005

Table 6 Mann–Whitney U test: rank table with respect to the independent variable "use of UML UCD"

UML vs Other Diagrams	N	Mean Rank	Sum of Ranks
UndQ1			
1	11	10.00	110.00
2	9	11.11	100.00
Total	20		
UndQ2			
1	11	10.50	115.50
2	9	10.50	94.50
Total	20		
UndQ3			
1	11	8.73	96.00
2	9	12.67	114.00
Total	20		
LQ1			
1	11	9.09	100.00
2	9	12.22	110.00
Total	20		
UQ1			
1	11	10.36	114.00
2	9	10.67	96.00
Total	20		
UQ2			
1	11	9.68	106.50
2	9	11.50	103.50
Total	20		
UQ4			
1	11	9.55	105.00
2	9	11.67	105.00
Total	20		
UQ5			
1	11	10.18	112.00
2	9	10.89	98.00
Total	20		

Table 7 depicts the results of Mann–Whitney U test for "Perceived Understandability 2" hypothesis. The significance values for these three questions (i.e., 0.659, 1.0, and 0.1 are all greater than $\alpha = 0.05$) show that there is no significant difference between the group of participants who used UML UCD models and the group of participants who used different types of diagrams/models, with respect to the perceived understandability. Thus, we accept the null hypothesis "H0-perceived-understandability-2."

7.4.3 Analysis of the perceived learnability data

Question LQ1 aims to assess the perceived learnability of the technique. Most of the respondents believe that they can learn the new technique quickly (79.3% responded with either 'agree' (51.72%) or 'strongly agree' (27.59%)), while only 20.7% were neutral.

In addition, in order to test hypotheses "Perceived Learnability 1" and "Perceived Learnability 2," we have conducted Mann–Whitney U test [72] on responses to question LQ1. Table 5 depicts the results of Mann–Whitney U test for "Perceived Learnability 1" hypothesis. The significance value (i.e., Asym. Sig. (2-tailed)) for LQ1 (i.e., 0.05 is equal to $\alpha = 0.05$) shows that there is a significant difference between the group of participants using diagrams/models and the group of participants not using diagrams/models, with respect to the perceived learnability. Thus, we reject the null hypothesis "H0-perceived-learnability-1" and accept the alternative hypothesis "H1-perceived-learnability-1."

Table 7 depicts the results of Mann–Whitney U test for "Perceived Learnability 2" hypothesis. The significance values for LQ1 (i.e., 0.179 is greater than $\alpha = 0.05$) shows that there is no significant difference between the group of participants who used UML UCD models and the group of participants who used different types of diagrams/models, with respect to the perceived learnability. Thus, we accept the null hypothesis "H0-perceived-learnability-2."

Table 7 Mann–Whitney U test results with respect to the independent variable "use of UML UCD"

	UndQ1	UndQ2	UndQ3	LQ1	UQ1	UQ2	UQ4	UQ5
Mann–Whitney U	44.000	49.500	30.000	34.000	48.000	40.500	39.000	46.000
Wilcoxon W	110.000	94.500	96.000	100.000	114.000	106.500	105.000	112.000
Z	− .441	.000	− 1.644	− 1.345	− .122	− .807	− .852	− .296
Asymp. Sig. (2-tailed)	.659	1.000	.100	.179	.903	.420	.394	.767

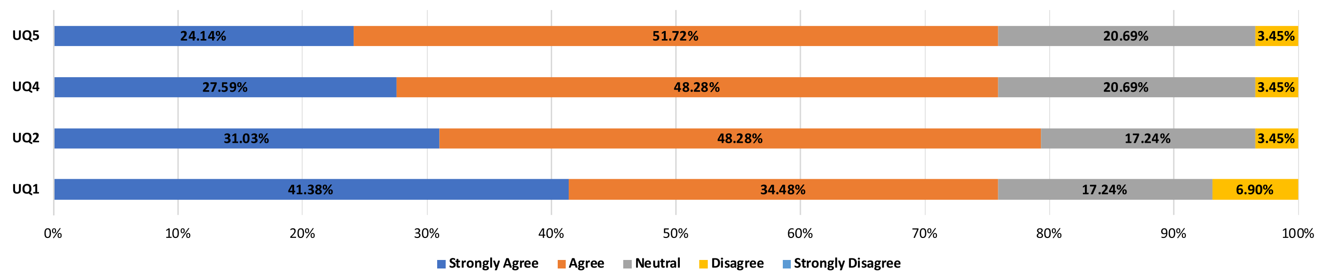


Fig. 22 Analysis of perceived usefulness data

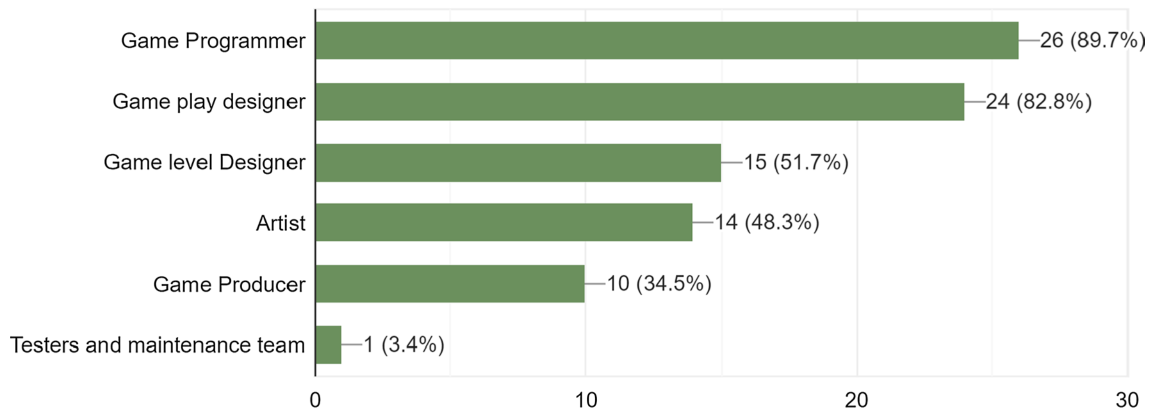


Fig. 23 Analysis of UQ3 data

7.4.4 Analysis of the perceived usefulness

Question UQ1 aims to assess the perceived usefulness of the graphical notation in helping game development teams to be more effective. Results (see Fig. 22) show that 75.9% of the participants agreed (41.4% strongly agreed and 34.5% agreed) with the statement. Only 6.9% of participants disagreed with the statement, while 17.2% have chosen the neutral option.

Question UQ2 aims to assess whether the use case textual description is informative. Results show that almost 80% of the participants found that the textual description is informative (48.3% agreed and 31% strongly agreed). Only one participant disagreed.

Question UQ4 aims to assess whether the proposed technique enhances the communication of game requirements with the development team. Results show that 75.9% agreed (27.6% 'strongly agree' and 48.3% 'agree') with the statement, while 20.7% were neutral. Only one disagreed with the statement.

Question UQ5 aims to assess the potential impact of the technique on team productivity. Results show that 75.8% agreed (24.1% 'strongly agree' and 51.7% 'agree') with the statement, while 20.7% were neutral. Only one participant disagreed with the statement.

The proposed approach can be useful for many roles within the game development team. Question UQ3 aims to know which roles would benefit the most from this technique. Responses collected from this question are captured using **vUQ3** variable. Figure 23 shows that the top roles are game programmer (selected by 26 participants) and game play designer (selected by 24 participants), followed by game level designer (selected by 15 participants), artists (selected by 14 participants), and game producer (selected by 10 participants). Only one participant thought that this technique would be useful for testers and maintenance teams.

In addition, in order to test hypotheses "Perceived Usefulness 1" and "Perceived Usefulness 2," we have conducted Mann–Whitney U test [72] on responses to questions UQ1, UQ2, UQ4, and UQ5. Table 5 depicts the results of Mann–Whitney U test for "Perceived Usefulness 1" hypothesis. The significance values (i.e., Asym. Sig. (2-tailed)) for these four questions (i.e., 0.029, 0.003, 0.044, and 0.005 are all less than $\alpha = 0.05$) show that there is a significant difference between the group of participants using diagrams/models and the group of participants not using diagrams/models, with respect to the perceived usefulness. Thus, we reject the null hypothesis "H0-perceived-usefulness-1" and accept the alternative hypothesis "H1-perceived-usefulness-1."

Table 7 depicts the results of Mann–Whitney U test for "Perceived Usefulness 2" hypothesis. The significance values for these four questions (i.e., 0.903, 0.42, 0.394, and 0.767 are all greater than $\alpha = 0.05$) show that there is no significant difference between the group of participants who used UML UCD models and the group of participants who used different types of diagrams/models, with respect to the perceived usefulness. Thus, we accept the null hypothesis "H0-perceived-usefulness-2."

7.4.5 Analysis of the suggested improvements

This question is optional, thus not all participants provided their opinions. In what follows, we summarize the significant suggestions, and we provide our feedback for each:

- *Suggestion 1* Support functional game use cases with test scenarios to improve the model understanding. *Response 1* Each game textual use case representation can be converted to a test scenario. For interdependent use case (i.e., interconnected through includes/extends relationships), their textual flows can be merged to create a more inclusive test scenario. In addition, these test scenarios may help validate the model correctness and completeness (covering all alternative paths).
- *Suggestion 2* Extend the technique by adding mechanisms to support iterative development. *Response 2* An iterative development approach aims to address certain types of risks sooner by implementing and integrating risky functionalities (e.g., poorly understood functionalities) earlier in the process. In the context of game development, some aspects are more important than others, e.g., such as the functions and behaviors of a game [73], represented as *Function* game use case in our approach. Starting with these crucial game aspects would allow for early analysis. Hence, they can be queried using a wizard in the first steps of the creation of the UCD.
- *Suggestion 3* Add more details on how certain mechanics and functionalities work. *Response 3* The user may add such details to the textual description of use cases.
- *Suggestion 4* Create some models for the most important game mechanics that can be used in future projects. *Response 4* This suggestion can be addressed by adding some predefined game mechanics (as stored templates) that game designer can reuse.
- *Suggestion 5* Extend the technique to allow for task distribution between team members. *Response 5* Task distribution is relevant to how the project is being managed. However, it may be addressed by adding some fields to the textural description of each use case, e.g., a text field specifying the owner of the use case.
- *Suggestion 6* Supporting more visuals in the diagram will help improve the understandability, such as adding colors for each depth level in the diagram, or colors for each type of game use cases. *Response 6* Visual variables, such as size, color, and location of symbols, have properties that make them suitable for encoding different types of information [74]. In addition, *color* is one of the most cognitively effective of all visual variables [74]. However, augmenting the notation with more visuals and colors may increase the graphic complexity, which we aim to keep cognitively manageable. Hence, a study of the impact of such additions from syntactic, semantic, and cognitive aspects is required.

All listed suggestions are interesting and represent a good pool of features to select from for our future GUCM releases.

7.5 Discussion

In what follows, we discuss the findings presented in Sect. 7.4. The results of our first question CQ1 show that most of our survey participants are game programmers (16 out of 29). While this may seem biased to some readers, it is not surprising that game programmers account for a huge proportion of employees at any given game studio. Although there is a strong evidence from the literature [5, 8] that the use of requirements models has a positive impact on game development processes by solving the communication issues between stakeholders with technical and art backgrounds [4], they are not widely used in the game industry. In our study, this fact was observed when analyzing responses to question CQ4, where only 31% of respondents have used UML use case diagrams in their work. Hence, there is a need to investigate and address the problems that are preventing the adoption of such techniques within the game industry.

Responses to question CQ5 confirm the findings of Callele et al. [4] and Dormans [21] with respect to the existence of communication and understandability issues of game requirements among stakeholders. Our proposed technique aims to help solve such obstacles by endorsing the use of UC diagrams in eliciting game requirements. Furthermore, practitioners using modeling notations are primarily interested in its low learning curve [75]. Hence, we took this aspect into consideration by keeping our proposed approach as simple as possible in order to make it more accessible to non-technical personals and facilitate its adoption. The analysis of the perceived understandability and learnability data (i.e., responses to question UndQ1, UndQ2, UndQ3 for the perceived understandability and LQ1 for the perceived learnability) confirms this fact.

In addition, even though all participants provided a positive assessment of the perceived understandability, learnability, and usefulness of the proposed approach (as discussed in Sects. 7.4.2, 7.4.3, and 7.4.4), results of Mann–Whitney U test indicate that there is a significant difference in the results between the group of participants having prior experience with diagrams/models and the group of participants who are not familiar with diagrams/models. More specifically, the approach appears to be more appealing (may be compared to the commonly used natural language requirements) for participants who are not familiar with diagrams. Although unexpected, this finding may be due to the fact that we have provided participants with a very short and simple presentation of our approach. A more formal training on the proposed approach and more complex examples may have unveiled some modeling difficulties (that the participants who are familiar with models are aware of) and may have led to different results.

Furthermore, we notice that there is a no significant difference between the group of participants having prior knowledge of UML use case diagrams and the group of participants who used other types of diagrams/models, with respect to the perceived understandability, learnability, and usefulness of the proposed approach. This finding was expected, as use case diagrams are in general simple (e.g., limited number of constructs) compared to other modeling notations. Hence, this is a positive indication towards the acceptance and potential adoption of our approach by game practitioners who are familiar with requirements modeling.

Furthermore, the results analysis of the perceived usefulness of the proposed approach is very promising, with game programmers, game play designers, and game level designers, being the top three roles that would benefit the most from the proposed technique. These roles represent a very large proportion of game development industry.

8 Threats to validity

The proposed approach, the GUCM prototype tool, the illustrative examples, and the empirical evaluation are subject to several limitations and threats to validity, categorized here according to three important types identified by Wright et al. [76].

In terms of *construct validity*, a potential threat may be related to the fact that the survey participants might have had a different understanding of the questions than what we had intended. To mitigate that threat, we tried to make the questions as simple as possible, using concise terms. To assess the perceived understandability, learnability, and usefulness, we have used closed-ended questions (preempting a particular answer) to collect participants' self-assessment about our proposed technique, which may be subjective and biased. In order to better assess understandability, learnability, and usefulness, we plan to conduct a controlled experiment involving design tasks, observations, and measurements. Moreover, another potential threat is that the material given to the respondents (description of the main constructs of the approach and one illustrative example) might not be enough to ensure that participants fully understand the approach. In addition, participants were not able to ask questions, since the experiment was conducted online. To reduce this risk, we have selected Super Mario Bros, a well-known game within the gaming community, as an illustrative example. However, using not well-known game examples may lead to different responses. Another possible threat is related to the scalability of the proposed technique. Our technique adds new game-related kinds of actors and use cases to the UML use case diagram notation, which already suffers from the lack of scalability [77]. To mitigate this threat, modelers can manage the complexity of the produced use case models via modularization [78]. Modularization involves the partitioning of the use case model into use case packages. A use case package is used to structure the use case model by dividing it into small chunks. However, applying modularization is out of the scope of this study.

Regarding *internal validity*, a possible threat is that the proposed UML use case meta-model extensions might have missed some game-related aspects. To mitigate this threat, we opted for a high level abstraction when defining the additional types, which would hide many details related to actual implementations. For example, security functions and multiplayer management may be described using ≪Function≫ use cases, whereas playing music, playing sound effects may be described using ≪PlayAudio≫ use cases. Moreover, we wanted to keep the meta-model as simple as possible to promote adoption and to help future extensions without the need for major meta-model restructuring. Another potential threat may be related to the usefulness and potential adoption of the proposed use case-based game modeling technique in real game development life cycle. To reduce this risk, we have explored the acceptability of the approach by conducting a survey among game developers. Results have shown that most participants found the approach useful. Another possible threat is that the survey participants may be working for one or two companies, which would have an impact on our results. To mitigate this risk, the link to our survey was published within the game communities on Twitter, known for their diversity, i.e., different background, locations, companies, etc.

In terms of *external validity*, there is a concern with respect to the applicability of the proposed game-oriented use case modeling approach and the GUCM tool, to multiple game genres. To mitigate this threat, we have applied the approach to four games of different genres. Indeed, Super Mario Bros (see Sect. 6.1) is a platformer, Tetris (see Sect. 6.2) is of arcade/puzzle type, Just Dance (see Sect. 6.3) is a motion-based rhythm game, and The Walking Dead (see Sect. 6.3) is an episodic adventure interactive drama game. As future work, we are planning to apply our approach to other game genres. Another threat is related to ensuring that the results drawn from the survey can be generalized to the entire population of game development community. To maximize the external validity, we did our best to reach as many different profiles of the gaming population as possible, by utilizing Twitter's popular active hashtags used by people from the game development industry on the platform. Indeed, Twitter is considered as the best social media platform for game developers when it comes to building communities and communications.[3],[4] In addition, #gamedev and #indiedev hashtags are often ranked in the top 10 most active game development hashtags[5], hence they were used to target game developers. Another potential threat is related to the relatively small number of participants in our study and the limited information we have collected about their demographics (e.g., age, country of origin, etc.), which may affect our ability to generalize our findings.

9 Conclusions and future work

We have introduced a novel use case-based game modeling approach. The proposed technique extends the standard UML use case Model, to allow for better description of game-related requirements and for promoting a common understanding of game requirements among game development teams. As a proof of concept, we have developed a web based prototype tool called GUCM (game use case modeling). The proposed approach and tool were validated using four well-known games, Super Mario Bros, Tetris, Just Dance, and The Walking Dead. Furthermore, to assess the understandability and the usefulness of the proposed approach, we have conducted a survey within the game development community on Twitter. Results indicated an agreement about the added value of the proposed approach and a willingness of adoption by the game development community.

As future work, we plan to improve the visuals of the use case diagram by introducing coloring schema. Furthermore, we plan to conduct a controlled experiment in order to measure the usability and learnability of the proposed approach and tool.

Appendix

See Table 8.

Acknowledgements The authors would like to acknowledge the support provided by the Deanship of Scientific Research at King Fahd University of Petroleum & Minerals for funding this work through Project No. SB191011.

[3] https://indieboost.com/blog/how-to-effectively-use-social-media-as-an-indie-game-developer/.

[4] https://www.gamemarketinggenie.com/blog/social-media-platforms-for-gamers.

[5] https://ritetag.com/best-hashtags-for/gamedev.

Table 8 Summary of the perceived understandability/learnability/usefulness responses

	Survey Questions							
	UndQ1	UndQ2	UndQ3	LQ1	UQ1	UQ2	UQ4	UQ5
Respondents who have not used diagrams								
1	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree
2	Strongly agree	Strongly agree	Strongly agree	Agree	Agree	Strongly agree	Agree	Agree
3	Agree	Agree	Strongly agree	Neutral	Disagree	Strongly agree	Agree	Agree
4	Agree	Agree	Agree	Agree	Strongly agree	Agree	Agree	Strongly agree
5	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree
6	Strongly agree	Strongly agree	Agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree
7	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Agree	Agree
8	Neutral	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree
9	Agree	Neutral	Neutral	Agree	Strongly agree	Neutral	Agree	Agree
Respondents who have used diagrams other than UML								
1	Disagree	Agree	Strongly agree	Neutral	Strongly agree	Agree	Agree	Agree
2	Agree	Strongly agree	Agree	Agree	Agree	Agree	Agree	Agree
3	Agree	Agree	Neutral	Strongly agree	Strongly agree	Agree	Strongly agree	Strongly agree
4	Agree	Agree	Neutral	Agree	Agree	Agree	Neutral	Agree
5	Strongly agree	Agree	Agree	Agree	Strongly agree	Agree	Agree	Agree
6	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral	Agree	Neutral
7	Strongly Disagree	Neutral	Agree	Agree	Strongly agree	Neutral	Agree	Agree
8	Disagree	Agree	Strongly agree	Neutral	Neutral	Agree	Disagree	Disagree
9	Agree	Agree	Neutral	Neutral	Neutral	Neutral	Neutral	Neutral
10	Strongly agree	Strongly agree	Neutral	Agree	Neutral	Agree	Agree	Agree
11	Agree	Agree	Agree	Agree	Neutral	Agree	Neutral	Neutral
Respondents who have used UML								
1	Neutral	Neutral	Agree	Agree	Agree	Disagree	Agree	Agree
2	Disagree	Neutral	Agree	Neutral	Agree	Agree	Strongly agree	Agree
3	Strongly agree	Agree	Agree	Agree	Agree	Agree	Agree	Agree
4	Agree	Agree	Agree	Agree	Agree	Agree	Agree	Neutral
5	Agree	Strongly agree	Strongly agree	Strongly agree	Agree	Strongly agree	Strongly agree	Strongly agree
6	Strongly agree	Agree	Agree	Agree	Disagree	Neutral	Neutral	Neutral
7	Agree	Agree	Agree	Agree	Agree	Agree	Neutral	Agree
8	Agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Strongly agree	Agree
9	Neutral	Agree	Agree	Agree	Agree	Agree	Neutral	Neutral

References

1. GamesIndustry.biz. Global games market value rising to $134.9bn in 2018. https://www.gamesindustry.biz/articles/2018-12-18-global-games-market-value-rose-to-usd134-9bn-in-2018. Accessed Feb 2020
2. Planet Market Reports. Global games market value rising to $175.9bn in 2025. https://www.rednewswire.com/global-games-market-value-rising-to-175-9bn-in-2025/. Accessed Feb 2020
3. Blow J (2004) Game development: harder than you think. Queue 1(10):28–37
4. Callele D, Neufeld E, Schneider KA (2005) Requirements engineering and the creative process in the video game industry. In: 13th IEEE international conference on requirements engineering (RE 2005), 29 August-2 September 2005. France. IEEE Computer Society, Paris, pp 240–252
5. Kasurinen J, Maglyas A, Smolander K (2014) Is requirements engineering useless in game development? In: Salinesi C, van de Weerd I (eds) Requirements engineering: foundation for software quality—20th international working conference, REFSQ 2014, Essen, Germany, April 7–10, 2014. Proceedings, volume 8396 of lecture notes in computer science. Springer, pp 1–16
6. Washburn Jr. M, Sathiyanarayanan P, Nagappan M, Zimmermann T, Bird C (2016) What went right and what went wrong: an analysis of 155 postmortems from game development. In: Dillon LK, Visser W, Williams LA (eds) Proceedings of the 38th international conference on software engineering, ICSE 2016, Austin, TX, USA, May 14–22, 2016—companion volume. ACM, pp 280–289
7. Hussain A, Asadi O, Richardson DJ (2018) A holistic look at requirements engineering practices in the gaming industry. CoRR, arXiv:1811.03482
8. Reyno EM, Cubel JÁC (2008) Model driven game development: 2d platform game prototyping. In: Botti VJ, Barella A, Carrascosa C (eds) GAMEON'2008, (covers game methodology, game graphics, AI behaviour, game AI analysis, AI programming, neural networks and agent based simulation, team building, education and

social networks), November 17–19, 2008. UPV, Valencia, Spain, EUROSIS, pp 5–7

9. Flood K (2003) Game unified process. https://www.gamedev.net/articles/programming/general-and-gameplay-programming/game-unified-process-r1940/
10. Koutonen J, Leppänen M (2013) How are agile methods and practices deployed in video game development? A survey into finnish game studios. In: Baumeister H, Weber B (eds) Agile processes in software engineering and extreme programming—14th international conference, XP 2013, Vienna, Austria, June 3–7, 2013. Proceedings, volume 149 of lecture notes in business information processing. Springer, pp 135–149
11. Petrillo F, Pimenta MS (2010) Is agility out there? Agile practices in game development. In: Anacleto JC, de Mattos Fortes RP, Costa CJ (eds) Proceedings of the 28th annual international conference on design of communication, SIGDOC 2010, São Carlos, São Paulo state, Brazil, September 26–29, 2010. ACM, pp 9–15
12. Djaouti D, Alvarez J, Jessel J-P, Methel G (2008) A gameplay definition through videogame classification. Int J Comput Games Technol 2008:70350:1-470350:7
13. Ollila EMI, Suomela R, Holopainen J (2008) Using prototypes in early pervasive game development. Comput Entertain 6(2):17:1-17:17
14. Coram M, Bohner SA (2005) The impact of agile methods on software project management. In: 12th IEEE international conference on the engineering of computer-based systems (ECBS 2005), 4–7 April 2005, Greenbelt, MD, USA. IEEE Computer Society, pp 363–370
15. McKenzie T, Trujillo MM, Hoermann S (2019) Software engineering practices and methods in the game development industry. In: Extended abstracts of the annual symposium on computer-human interaction in play companion extended abstracts, CHI PLAY—19 extended abstracts. Association for Computing Machinery, New York, pp 181–193
16. Godoy A, Barbosa EF (2010) Game-scrum: an approach to agile game development. In: Proceedings of SBGames. pp 292–295
17. Gonzalez-Salazar M, Mitre-Hernandez H, Lara-Alvarez C (2017) Method for game development driven by user-experience: a study of rework, productivity and complexity of use. Int J Adv Comput Sci Appl 8(2):394–402
18. Folmer E (2007) Component based game development–a solution to escalating costs and expanding deadlines? In: Schmidt HW, Crnkovic I, Heineman GT, Stafford JA (eds) Component-based software engineering, 10th international symposium, CBSE 2007, Medford, MA, USA, July 9–11, 2007, proceedings, volume of 4608 lecture notes in computer science. Springer, pp 66–73
19. Brambilla M, Cabot J, Wimmer M (2017) Model-driven software engineering in practice. Synthesis lectures on software engineering, 2nd edn. Morgan & Claypool Publishers, New England
20. Loniewski G, Insfran E, Abrahão S (2010) A systematic review of the use of requirements engineering techniques in model-driven development. In: International conference on model driven engineering languages and systems. Springer, pp 213–227
21. Dormans J (2012) The effectiveness and efficiency of model driven game design. In Herrlich M, Malaka R, Masuch M (eds) Entertainment computing—ICEC 2012—11th international conference, ICEC 2012, Bremen, Germany, September 26–29, 2012. Proceedings, volume 7522 of lecture notes in computer science. Springer, pp 542–548
22. Petrillo F, Pimenta M, Trindade F, Dietrich C (2009) What went wrong? A survey of problems in game development. Comput Entertain 7(1):13
23. Callele D, Neufeld E, Schneider KA (2006) Emotional requirements in video games. In: 14th IEEE international conference on requirements engineering (RE 2006), 11–15 September 2006, Minneapolis/St.Paul, Minnesota, USA. IEEE Computer Society, pp 292–295
24. Paschali ME, Ampatzoglou A, Chatzigeorgiou A, Stamelos I (2014) Non-functional requirements that influence gaming experience: a survey on gamers satisfaction factors. In: Proceedings of the 18th international academic MindTrek conference: media business, management, content & services. ACM, pp 208–215
25. Cheng BHC, Atlee JM (2007) Research directions in requirements engineering. In Briand LC, Wolf AL (eds) International conference on software engineering, ISCE 2007, workshop on the future of software engineering, FOSE 2007, May 23–25, 2007, Minneapolis, MN, USA. IEEE Computer Society, pp 285–303
26. OMG (2017) OMG Unified Modeling Language—version 2.5.1. https://www.omg.org/spec/UML/2.5.1. Accessed Dec 2020
27. Wolf MJP (2001) Genre and the video game. The medium of the video game. pp 113–134
28. Wikipedia. Super Mario Bros. https://en.wikipedia.org/wiki/Super_Mario_Bros. Accessed Dec 2020
29. Wikipedia. Tetris. https://en.wikipedia.org/wiki/Tetris. Accessed Dec 2020
30. Wikipedia. Just dance. https://en.wikipedia.org/wiki/Just_Dance_(video_game). Accessed Dec 2020
31. Wikipedia. The walking dead. https://en.wikipedia.org/wiki/The_Walking_Dead_(video_game). Accessed Dec 2020
32. Ambler SW (2001) Agile modeling: a brief overview. In: Evans A, France RB, Moreira AMD, Rumpe B (eds) Practical UML-based rigorous development methods—countering or integrating the eXtremists, workshop of the pUML-group held together with the "UML" 2001, October 1st, 2001 in Toronto, Canada, volume P-7 of LNI. GI, pp 7–11
33. Ambler SW (2002) Agile modeling: effective practices for extreme programming and the unified process. Wiley, New Jersey
34. Chaudron MRV, Werner H, Ariadi N (2012) How effective is UML modeling? Softw Syst Model 11(4):571–580
35. Misbhauddin M, Alshayeb M (2015) Extending the UML use case metamodel with behavioral information to facilitate model analysis and interchange. Softw Syst Model 14(2):813–838
36. Sauer S, Engels G (2001) Uml-based behavior specification of interactive multimedia applications. In: 2002 IEEE CS international symposium on human-centric computing languages and environments (HCC 2001), September 5–7, 2001 Stresa, Italy. IEEE Computer Society, pp 248–255
37. Fowler M, Scott K (2000) UML distilled—a brief guide to the Standard Object Modeling Language. notThenot Addison-Wesley object technology series, 2nd edn. Addison-Wesley-Longman, Boston
38. Cockburn A (2000) Writing effective use cases, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston
39. OMG (2013) OMG Meta Object Facility (MOF) Core Specification. Version 2.4.1
40. Zhu M, Wang AI (2019) Model-driven game development: a literature review. ACM Comput Surv 52(6):123
41. Novak J (2011) Game development essentials: an introduction. Cengage Learning, Boston
42. Tang S, Hanneghan M, Hughes T, Dennett C, Cooper S, Ariff Sabri M et al. (2008) Towards a domain specific modelling language for serious game design. In: 6th international game design and technology workshop, Liverpool, UK
43. Tang S, Hanneghan M, Carter C (2013) A platform independent game technology model for model driven serious games development. Electron J e-Learn 11(1):61–79
44. Erickson J, Siau K (2007) Can uml be simplified? practitioner use of uml in separate domains. In: Proceedings EMMSAD, vol 7. pp 87–96. Citeseer
45. Hernandez FE, Ortega FR (2010) Eberos gml2d: a graphical domain-specific language for modeling 2d video games. In:

Proceedings of the 10th workshop on domain-specific modeling. pp 1. Citeseer

46. Herzig P, Jugel K, Momm C, Ameling M, Schill A (2013) Gaml-a modeling language for gamification. In: 2013 IEEE/ACM 6th international conference on utility and cloud computing (UCC). IEEE, pp 494–499
47. Reyno EM, Cubel JÁC (2009) Automatic prototyping in model-driven game development. Comput Entertain 7(2):29
48. Pleuss A, Hussmann H (2011) Model-driven development of interactive multimedia applications with MML. In: Hussmann H, Meixner G, Zuehlke D (eds) Model-driven development of advanced user interfaces. Springer, pp 199–218
49. de Lope RP, Medina-Medina N (2016) Using UML to model educational games. In: 8th international conference on games and virtual worlds for serious applications, VS-GAMES 2016, Barcelona, Spain, September 7–9, 2016. IEEE Computer Society, pp 1–4
50. Hog CE, Djemaa RB, Amous I (2011) Towards an UML based modeling language to design adaptive web services. In: Proceedings of the international conference on semantic web and web services. pp 38–44
51. Murali R, Ireland A, Grov G (2015) A rigorous approach to combining use case modelling and accident scenarios. In: Havelund K, Holzmann GJ, Joshi R (eds) NASA formal methods–7th international symposium, NFM 2015, Pasadena, CA, USA, April 27–29, 2015, proceedings, volume 9058 of lecture notes in computer science. Springer, pp 263–278
52. Al-alshuhai A, Siewe F (2015) An extension of the use case diagram to model context-aware applications. In: 2015 SAI intelligent systems conference (IntelliSys). IEEE, pp 884–888
53. Yue T, Zhang H, Ali S, Liu C (2016) A practical use case modeling approach to specify crosscutting concerns. In: Kapitsaki GM, de Almeida ES (eds) Software reuse: bridging with social-awareness–15th international conference, ICSR 2016, Limassol, Cyprus, June 5–7, 2016, proceedings, volume 9679 of lecture notes in computer science. Springer, pp 89–105
54. Mai PX, Goknil A, Shar LK, Pastore F, Briand LC, Shaame S (2018) Modeling security and privacy requirements: a use case-driven approach. Inf Softw Technol 100:165–182
55. Cooper KML, Longstreet CS (2012) Towards model-driven game engineering for serious educational games: tailored use cases for game requirements. In: Mehdi QH, Elmaghraby A, Marshall I, Moreton R, Ragade RK, Zapirain BG, Chariker J, El-Said MM, Yampolskiy RV, Zhigiang NL (eds) 17th international conference on computer games, CGAMES 2012, Louisville, KY, USA, July 30–Aug. 1, 2012. IEEE Computer Society, pp 208–212
56. Jacobson I (2016) Use-case 2.0. Commun ACM 59(5):61–69
57. Aleem S, Capretz LF, Ahmed F (2016) Game development software engineering process life cycle: a systematic review. J Softw Eng Res Dev 4(1):6
58. Sonic the hedgehog, Sep 2019
59. Cockburn A (2000) Writing effective use cases. Addison-Wesley Professional, Boston
60. Bauer B, Odell J (2005) UML 2.0 and agents: how to build agent-based systems with the new UML standard. Eng Appl Artif Intell 18(2):141–157
61. Araujo Guedes GT, Vicari RM (2009) Applying AUML and UML 2 in the multi-agent systems project. In: Heuser CA, Pernul G (eds) Advances in conceptual modeling—challenging perspectives, ER 2009 workshops CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS, Gramado, Brazil, November 9–12, 2009. Proceedings, volume 5833 of lecture notes in computer science. Springer, pp 106–115
62. Johnson D, Wiles J (2003) Effective affective user interface design in games. Ergonomics 46(13–14):1332–1345
63. Cooper J (2019) Game anim: video game animation explained: a complete guide to video game animation. CRC Press, Boca Raton
64. Fassone R (2017) Every game is an Island: endings and extremities in video games. Bloomsbury Publishing USA, Boston
65. Unity Technologies. 2020. Unity. https://unity.com
66. Kitchenham BA, Pfleeger SL (2002) Principles of survey research: part 3: constructing a survey instrument. SIGSOFT Softw Eng Notes 27(2):20–24
67. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B (2012) Experimentation in software engineering. Springer, Berlin
68. International Standard Organization (ISO) (2001) International standard iso/iec 9126, information technology—product quality—part1: quality model
69. Michelsen CD, Dominick WD, Urban JE (1980) A methodology for the objective evaluation of the user/system interfaces of the MADAM system using software engineering principles. In Miles Jr. EP (ed) Proceedings of the 18th annual southeast regional conference, 1980, Tallahassee, Florida, USA, March 24–26, 1980. ACM, pp 103–109
70. Davis FD (1989) Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q 13(3):319–340
71. Chadha Rishi (2019) gaming on twitter
72. Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 18(1):50–60
73. Klemm C, Pieters W (2017) Game mechanics and technological mediation: an ethical perspective on the effects of mmorpg's. Ethics Inf Technol 19(2):81–93
74. Moody DL (2009) The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Trans Softw Eng 35(6):756–779
75. Ozkaya M (2018) Do the informal & formal software modeling notations satisfy practitioners for software architecture modeling? Inf Softw Technol 95:15–33
76. Wright HK, Kim M, Perry DE (2010) Validity concerns in software engineering research. In: FoSER. pp 411–414
77. El-Attar M (2019) Evaluating and empirically improving the visual syntax of use case diagrams. J Syst Softw 156:136–163
78. Baldwin CY, Clark KB (1999) Design rules: the power of modularity, vol 1. MIT Press, Cambridge