Bilkent University
Department of Computer Engineering

CS319 - Object Oriented Software Engineering Project

A Vampire's Past

Analysis Report

Tuğrulcan Elmas,
Hazal Yıldız,
Rıfat Cihan,
Merve Doğan

Progress/Analysis Report
October 27, 2015

Contents

Analysis Report

A Vampire's Past

1 Introduction

Adventure games are those games that players recklessly try to find items, use them on relevant objects or combine them to have progress. Those games caught people by their interesting stories, such as Monkey's Island. Nowadays, the game sector has many 3D adventure games. They either have horror theme in which player continuously run from the character, such as "Amnesia The Dark Descent" and "Outlast" or are story-oriented and the progress depend on people's decisions, such as "The Walking Dead". However, they lack the spirit of old school adventure games. Old adventure games had simple graphics and simple gameplay and but they were still good at both making players run and hide and get excited and providing a good story. "A Vampire's Past" will bring back this spirit. Apart from that, it will also be gothic-themed with a cool vampire character, which will be fun for vampire-fans.

2 Requirement Analysis

2.1 Overview

2.1.1 Story

A vampire's past is a 2D adventure game. It consists of several levels according to a storyline. At the beginning of every level, there are comic-style introductory pictures and texts to let the player know what's going on.

Figure 1 Introductory Pictures and Texts from Max Payne

The story is about a vampire called JohnDoe who does not remember how he became a vampire and searches for clues to find memories of his past. He does this by breaking into houses.

2.1.2 Levels

Every level is a 2D map of a house interior.

Figure 2 Example of an 2D House Interior From This War of Mine

In the map, there is the character which the player control, there are non-playable characters that the game controls and interactable objects such as tables and shelves which the player can interact with.

2.1.3 The Character

The player controls a character which is called JohnDoe. He can go left and right in the game, climb the ladders and go through doors. He has a blood bar which goes empty over time. He dies if it goes totally empty, then the game is over.. He can use items, he has an inventory that stores the items. He can interact with objects such as shelves and can take items from them. He must beware of the non-playable characters because he wants to be unseen. If one of them catches JohnDoe, the game is over.

Figure 3 Sprite of The Character

Figure 4 Blood Bar Sprite

2.1.4 Items and Interactable Objects

Figure 5 Blood Potion
Used for Refilling Blood

Figure 6 Diary, An Objective Item
Used for ending the level

Figure 7, Blood Freezer
Used for Decreasing Blood Decrement

Items exist in the game in order to make progress in the game or to use for the benefit of the character (such as refilling the blood bar.) Objective item is a special kind of item. Once the character have the item in his inventory, the level is finished and the player proceeds to the next level. The first level's objective item is the vampire's diary.

Figure 8 Various Interactable Objects From The Game, Taken From Neighbours From Hell

Interactable objects contain items. The player can take what's inside once in front of them. Some of them may be locked. If it is the case, the character should have the item "Key" in his inventory.

Figure 9 Character Inventory From Neighbours From Hell

The player hold items in his character's inventory. He can use them by pressing the appropriate number from the keyboard.

2.1.5 Non-Playable Characters

Figure 10 The Sleeping Man, From This War of Mine

The non-playable characters are controlled by people and have premade movement algorithm. The character's mission is to avoid being seen by them. Once he is seen, the game is over. Sleeping man is a non-playable character who generally sleeps but have a waking up frequency. He can also wake up if the character makes noise by moving constantly.

2.1.6 Ladders

Figure 11 An Example of A Ladder From This War of Mine

Ladders are used to transport the character between floors. Once in front of them, the character can interact with a ladder to switch between floors.

2.2 Functional Requirements

2.2.1. Playing A New Game

Figure 12 Main Menu, From Vampire Masquerade - Bloodlines

The player should be able to play the game from the beginning starting from a provided menu without missing any storyline.

2.2.2. To Continue

The player should be able to continue from the level which he has saved. However he should not continue from the instant he has quit the game because it may result in cheating and would make the game easier.

2.2.3 The Background Music

The game should provide a background music that fits into the theme of the game. The music should change every level.

2.2.4 Sound Effects

There should sound effects according to character's and non-playable character's actions.

2.2.5 To Change Brightness

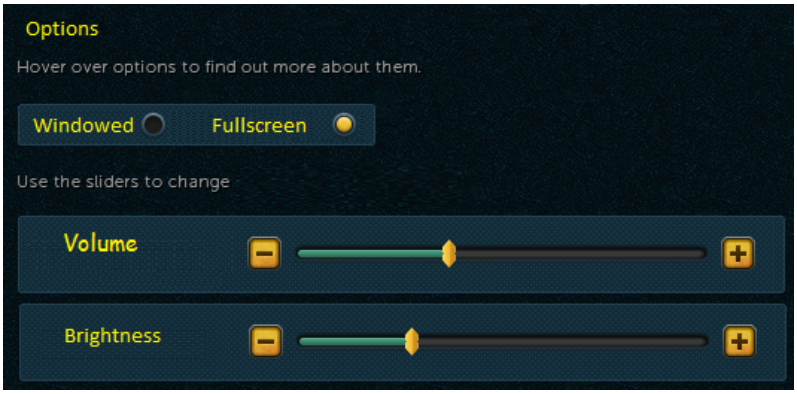


Figure 13 Options Screen

The player should be able to change the brightness of the screen, making it darker or brighter according to his desire. The game should update its brightness immeadiately as the player change the brightness so that the player will see how the screen looks like when he changed the brightness.

2.2.6 To Change Volume

The player should be able to change the volume of the sounds in the game.

2.2.7 To Go Fullscreen

The player should be able to play the game in full screen mode. He should also have the options to go back to window mode.

2.2.8 To Get Help

Figure 14 Help Screen

The game should provide controls to play the game and define the current level's objective as well.

2.2.9 To Pause In Game

Figure 15 Pause Game Menu or In-Game Menu, From Neon Ninja

The player should be able to pause game in the game and continue playing when he desires. There should be no action on the map when the game is paused, the blood-bar should stop decreasing and there should be no data-loss when the players wants to resume. Pausing the game should give options to the player such as going to options screen, seeing the help screen or quitting the game.

2.2.10 To Save The Current Level

The game should automatically save the current level so that the player should be able to continue from that level after he reenters the game.

2.3 Non-functional Requirements

2.3.1 Fluent Gameplay

Game's characters and non-playable characters should move fluently so that any observer of the game should believe that they are actually moving humans in the game. There should be no delay to the character's movement when the player presses a direction key to move him.

2.3.2 Portability

For end users who are not very proficient in using a computer, the game should require no installation and be portable so that it can easily copied and carried in a flashdrive.

2.3.3 Operating System Compatibility

The game should have versions in the popular platforms such as Windows, Linux, Mac, Android and IOS.

2.3.4 2D Graphics

The game should have 2D graphics in order to give the spirit of old 2D adventure games.

2.3.5 No Camera Movement

For the spirit of the old games and for the sake of simplicity, there should be no game camera. The map should be small enough to fit in the whole screen.

2.4 Constraints

2.4.1 English

In order attract players from all around the world, the game is constrained in English language. The other languages will be added as new features.

2.4.2 Open Source

We want our game to be an open source game so everybody will help us to add new features and build a more complex and an entertaining game.

2.5 Scenarios

2.5.1 New Game Success Scenario Part I The Player Starts The Game

1. Tuğrulcan clicks New Game button from the main menu.

2. Tuğrulcan sees vampire's picture with the first level's story texts.

3. Tuğrulcan sees game's map. It is a 2D house which has two floors. The first floor consist of only a corridor. The second floor has two rooms next to each other.

4. Tuğrulcan sees one table on the first floor left corner.

5. Tuğrulcan sees one ladder on the left of the first ladder.

6. Tuğrulcan sees a man sleeping on the bed in the room on the right of the second floor.

7. Tuğrulcan sees a shelf left to the sleeping man.

8. Tuğrulcan sees his character JohnDoe who is a vampire, in the first floor next to the outside door.

9. Tuğrulcan sees a blood bar on the right of the screen.

2.5.2 New Game Success Scenario Part II The Player Finishes The First Level

1. Tuğrulcan pushes left to move JohnDoe to the left, until he reaches in front of the table.

2. Tuğrulcan hits enter to interact with table. JohnDoe takes the key on the table and puts it into his inventory.

3. Tuğrulcan pushes right to move JohnDoe right until JohnDoe reaches in front of the ladder.

4. Tuğrulcan pushes up constantly and JohnDoe climbs the ladder.

5. Tuğrulcan pushes right to move JohnDoe right until JohnDoe reaches in front of the shelf.

6. Tuğrulcan hits enter to open the shelf with his key.

7. Tuğrulcan makes JohnDoe take the diary in the shelf and put it into his inventory.

8.Tuğrulcan hits "1" to make JohnDoe read the diary (which is labeled as 1) in the inventory.

9. Tuğrulcan sees a notification which says that he has finished the first level succesfully.

2.5.3 Blood Bar Goes Empty Scenario

Note: Takes place after the New Game Success Scenario Part I.

1. Tuğrulcan is still in the game.

2. Tuğrulcan does not move his player or do anything. He is away from keyboard.

3. The blood bar on the right of the screen constantly decreases.

4. The blood bar goes empty.

5. After coming back to keyboard, Tuğrulcan sees a notification which says that he has lost the game.

6. Tuğrulcan sees the main menu.

2.5.4 Man Catches Player Scenario

Note: Takes place after the 5th step of the New Game Success Scenario Part II.

1. Tuğrulcan sees that the sleeping man wakes up.

2. The sleeping man is too close to Tuğrulcan's character JohnDoe.

3. The sleeping man catches Tuğrulcan.

4. Tuğrulcan sees a notification which says that he has lost the game.

5. Tuğrulcan sees the main menu.

2.5.5 Pause Game See Help Scenario

Note: Takes place after the New Game Success Scenario Part I.

1. Tuğrulcan pushes Esc.

2. Tuğrulcan sees the in-game menu consists of "Resume", "Options", "Help", "Quit" buttons.

3. Tuğrulcan clicks Help button.

4. Tuğrulcan sees the Help screen.

5. Tuğrulcan pushes OK.

6. Tuğrulcan again sees the in-game menu.

7. Tuğrulcan clicks on Resume to continue playing.

2.5.6 Change Brightness Scenario

1. Tuğrulcan clicks Options from the main menu.

2. Tuğrulcan sees Options screen.

3. Tuğrulcan sees a couple of options in the screen such as "Brightness", "Volume", with a slider next to it and "FullScreen" and "Windowed" with radio buttons.

4. Tuğrulcan move the slider next to the "Change Brightness" to increase the brightness.

5. The screen gets brighter.

6. Tuğrulcan presses "Back".

7. Tuğrulcan sees main menu again.

Note: Continue Game Success scenario is same as New Game Success scenario except it loads introductory images, texts, map and interactable objects of another level from the database.

2.6 Use Case Models

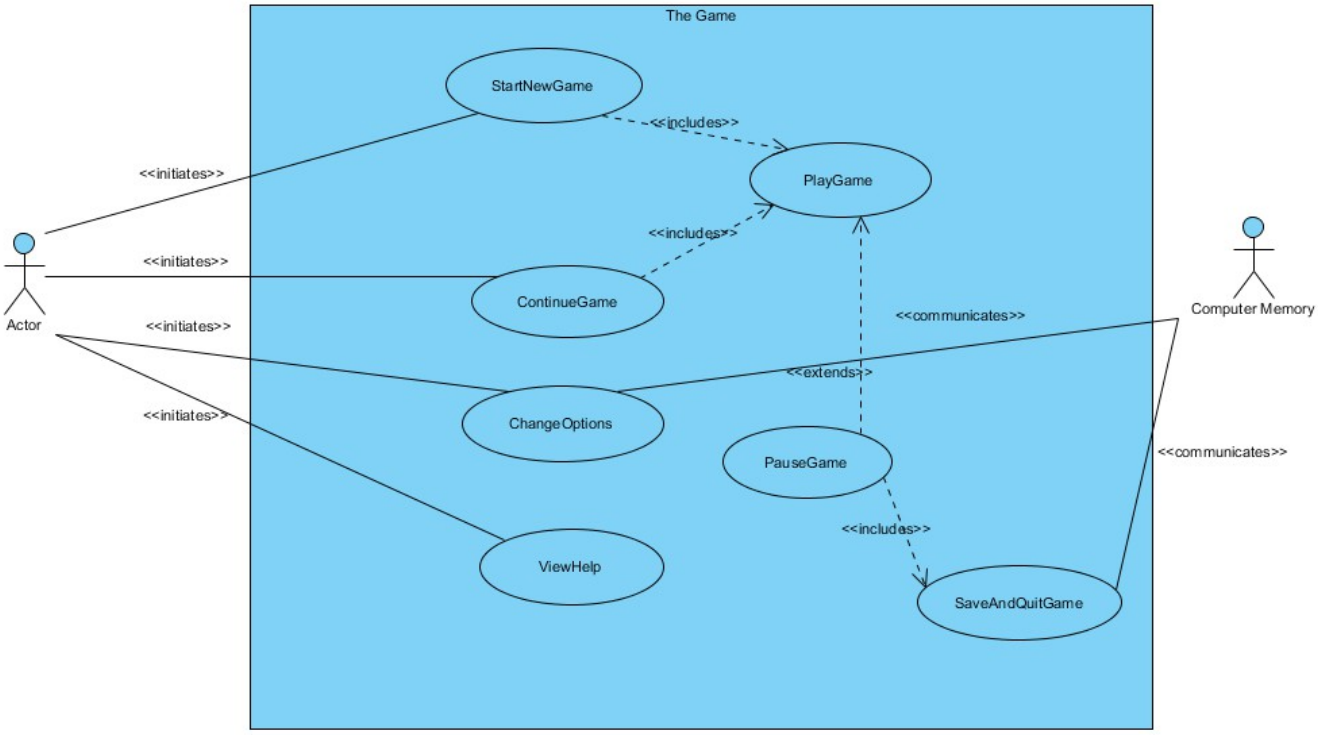


Figure 16 The Use Case Diagram

2.6.1 Start New Game Use Case

Use Case name: - StartNewGame

Participating actors: - Initiated by Player

Entry Conditions: - The player is in the game's main menu.

Flow of Events: - 1. The player clicks "New Game" from the main menu.

2. The game loads the first level of the game.

Exit Conditions: - The game loads the first level succesfully.

Post Conditions: - The game initiates PlayGame use case for the current level.

2.6.2 Continue Game Use Case

Use Case name: - ContinueGame

Participating actors: - Initiated by Player

- Communicates with The Computer Memory

Entry Conditions: - The player is in the main menu of the game.

- The player has saved the game before.

Flow of Events: - 1. The player clicks "Continue" from the main menu.

2. The game retains the last level the player has played from the memory of computer.

3. The game loads the level it has retained from the starting point.

Exit Conditions: - The game loads the last level succesfully.

Post Conditions: - The game initiates PlayGame use case for the current level.

2.6.3 Play Game Use Case

Use Case name: - PlayGame

Participating actors: - Initiated by Player

Entry Conditions: - Player selects "New Game" or "Continue" from the main menu or finishes the previous level.

Flow of Events: - 1. The Game displays the introductory texts and images of the current level.

2. The Player hits enter to proceed to the level.

3. The Game loads the current level and displays the map which is a 2D house.

4. The Player gains control of the character and moves it by pressing direction buttons.

5. The Game lets the character move if it is suitable to the map.

6. The Player hits enter to make his character interact with objects which is behind the character.

7. The Game puts the items to character's inventory if certain conditions are met.

8. The Player presses Esc to pause the game.

9. The Game initiates PauseGame use case.

10. The Player presses a number from the keyboard.

11. The Game consumes the corresponding item from the character's inventory and applies its benefits.

12. The Player moves around the sleeping man, who is a non-playable character.

13. The Game check noise level of the characters and decides if the sleeping man should wake up.

Extensions to 3

3.1 The Game displays the character in the place according to the story of the current level.

3.2 The Game plays the background music.

3.3 The Game displays a full blood bar.

3.4 The Game creates non-playable characters that moves, sleeps or stands still according to the story of the current level.

3.5 The Game displays character inventory in the bottom of the screen.

3.6 The blood bar stars to decrease over time.

Extensions to 4

4.1 The Player pushes right (or left).

4.2 The Game checks if the character does not collide with an object or the wall in the direction.

4.3.1 (If he does not collide) The game lets the character moves or makes it stop.

4.3.2 (If he collides) The game makes character stop.

4.4 The Player pushes up (or down).

4.5 The game checks if there is a ladder behind the character.

4.6.1 (If there is a ladder.) The game lets the character climb up/down the ladder

4.6.2 (If there is not a ladder.) The game makes the character stop.

Extensions to 6

6.1.1 The Player hits enter when in front of a table.

6.1.2 The game checks if there are items on the table.

6.1.3 (If there are items) The game puts items to the character’s inventory.

6.2.1 The Player hits enter when in front of a shelf which is locked.

6.2.2 The Game checks if the character has the item “Key” in his inventory.

6.2.3 (If he does) The Game puts items to the character’s inventory.

6.2.4 (If he does not) The Game notifies the player that the shelf is locked.

Extensions to 11

10.1.1 The Game removes the corresponding item from the inventory which is the blood potion.

10.1.2 The Game refills the blood bar by %25 percent.

10.2.1 The Game removes the corresponding item from the inventory which is the blood freezer.

10.2.2 The Game slows down the decrement frequency of the blood bar.

Extensions to 12

11.1.1 Player pushes direction keys constantly while the character is close to the sleeping man.

11.1.2 The noise level increases continuously and in big amount.

11.2.1 Player pushes direction keys in discrete times while the character is close to the sleeping man.

11.2.1 The noise level increases discretely and in small amount.

11.3 The Game checks if the noise is above the sleeping man's alertness.

11.4 (If it is) The Game wakes up the sleeping man.

Exit Conditions: - - The player takes the item which is the objective item of the current level or

- The player gets in the sight of a non-playable character who is awake.
- The blood bar gets totally empty.

2.6.4 Pause Game Use Case

Use case name: - PauseGame

Participating actors: - Initiated by Player

Entry Conditions: - The player is in the game.

Flow of events: - 1. The player pushes Esc from keyboard.

2.The game displays and hold still the in-game menu which consists of:

A. Resume

B. Save

C. Options

D. Help

E. Quit

A.1 The Player clicks on "Resume".

A.2 The Game resumes PlayGame use case.

B.1 The Player clicks on "Save".

B.2 The Game initiates SaveGame use case.

C.1 The Player clicks on "Options",

C.2 The Game initiates ChangeOptions use case.

D.1 The Player clicks on "Help",

D.2 The Game initiates GetHelp use case.

E.1 The Player clicks on "Quit",

E.2 The Game asks if player confirms the action.

E.3.1.1 The Player confirms.

E.3.1.2 The Game terminates the PlayGame use case and quits to the main menu.

E.3.2.1 The Player does not confirm.

E.3.2.2 The Game resumes PauseGame use case.

Exit conditions: The player clicks one of the options.

Post conditions: The game resumes or takes action according to initiated use case.

2.6.5 Save&Quit Game Use Case

Use Case name:	-	SaveAndQuitGame
Participating actors:	-	Initiated by Player
	-	Communicates with The Computer Memory
Entry Conditions:	-	The player is in the game.
	-	The player pauses the game.
Flow of Events:	-	1. The Player clicks "Quit" from the in-game menu.

2. The Game asks the player if he wants to save before quitting.

3.1.1 The Player clicks "Yes".

3.1.2 The Game deletes the old level that has been set for player to continue.

3.1.3 The Game saves the current level for player to continue after he quits the game to computer memory.

3.1.4 The Game displays a pop-up to let the player he has succesfully saved the game.

3.1.2 The Player clicks "No."

Exit Conditions: - The Game succesfully quits to the main menu.

2.6.6 Change Options Use Case

Use case name: - ChangeOptions

Participating actors: - Initiated by Player

Entry conditions: - The Player is in the main menu or paused the game and saw in in-game menu

Flow of events: - 1. Player clicks on Options button.

2. The game displays 4 different parameters for user to change.

A. Brightness

B. Background Music

C. Full screen

A. The player changes brightness..

B. The player changes volume of Background Music.

C. The player switches to Fullscreen or to Windowed mode.

3. The game applies the changes.

Exit conditions: - Player clicks on the "Back" button.

Post conditions: - The parameters are updated.

- The player resumes the menu which he clicked Options from.

2.6.7 View Help

Use case name: - GetHelp

Participating actors: - Initiated by Player

Entry conditions: - The Player is in the main menu or paused the game and saw the in-game menu.

Flow of events: - 1. Player clicks on Help button from the main menu or from the in-game menu.

2. The game displays the help screen which is just an image.

Exit conditions: - Player clicks on the "Back" button.

Post conditions: - The player resumes the menu which he clicked Help from.

2.7 User Interface

We already covered the User Interface in Overview. What you see in this section will be the same graphics. Note that those graphics are taken from other games but they are chosen because they are the same as what we have in our minds. Our user interface will only be different in colour.

- Main Menu: (Help Section Will Be Added)

Figure 17 The Main Menu

- Options:

Figure 18 Options

- Help Screen:

Figure 19 Help Screen

- In-Game Menu (or Pause Game Menu):

Figure 20 In Game Menu

Note: In the first version of the game, controls, restart level and restart checkpoint will be exempted.

- The Main Game:

Figure 21 Main Game Screen

Note: This War of Mine is very similar to our game in user interface. There are bunch of interactable objects and non-playable charactters. It has even a blood bar that looks like a thermometer!

3 Analysis

3.1 Object Model

3.1.1 Domain Lexicon

The Game: The system, which has the control of almost everything that does not need input from the player.

Main Menu: The menu that appears immeadiately the game is started. Consists of the buttons "New Game", "Continue", "Options", "Help".

Main Game: The core game within the game (which is the system), in which the player moves his character and the game map, non-playable characters and interactable objects etc. are displayed. The term "Game" generally used for the main game through report but it is quite easy to understand that the reference is made to the Main Game by the context.

Level: Determines which main game the player currently players. The main game and its elements changes according to level.

Introduction: The text and images that are related to the story, displayed on the screen before the game map is displayed.

Player: The user of the system, who clicks on buttons in menus and moves the characters and make it interact with objects. Once in the main game, it can be used with chracter interchangeably, because the player does not control any other entity in the main game.

Character: The entity whom the player is in charge of. The player can make it move right and left by pressing the keys on the keyboard or climb the ladder or go through doors. It can also interact with objects and take the items they contain if they do.

Game Map: The main screen of the main game which displays the house's map, the chracter, non-playable characters and interactable objects, doors, ladders.

In Game Menu (or Pause Game Menu): is displayed when the player presses Esc, namely, pauses the game. Freezes the game. Consists of "Resume", "Options", "Help", "Quit".

Ladder: When the character is in front of it, lets him climb if the player pushes up continously.

Interactable Object: Upon interaction with the character, it gives the items they have if the premade conditions are satisfied.

Shelf: A special kind of an interactable object which requires the character has the corresponding key as a condition, to give its items.

Character Inventory: The place where stores the items the character has. Is displayed in the bottom of the Game Map.

Item: any object that is stored by interactable objects or the character inventory. Can be taken or used.

Objective Item: is a special kind of object that once the player acquires, the main game is finished and the level is changed, and the new main game is loaded with the new level.

Non-Playable Character: The characters which are controlled by the game. Does actions with a premade algorithm. Non-playable characters have alertness and blood level.

Alertness: The value which determines if the non-playable character can catch the character and end the game (with the help of a premade algorithm that also depends on character's distance to the non-playable character.)

BloodLevel: is owned by the character and the non-playable characters. Determines to what extend the character can survive the main game or he can steal blood from the non-playable character.

Sleeping Man: is a kind of non-playable character. It is viewed on Game Map as a man lying on a bed sleeping. It has a frequency of waking up that determines when he wakes up. While he is sleeping, its alertness is treated as zero, so he cannot catch the character.

Options: the screen in which the player can change volume and brightness and set fullscreen.

Help: the screen in which the player sees the commands and objectives.

3.1.2 Class Diagram

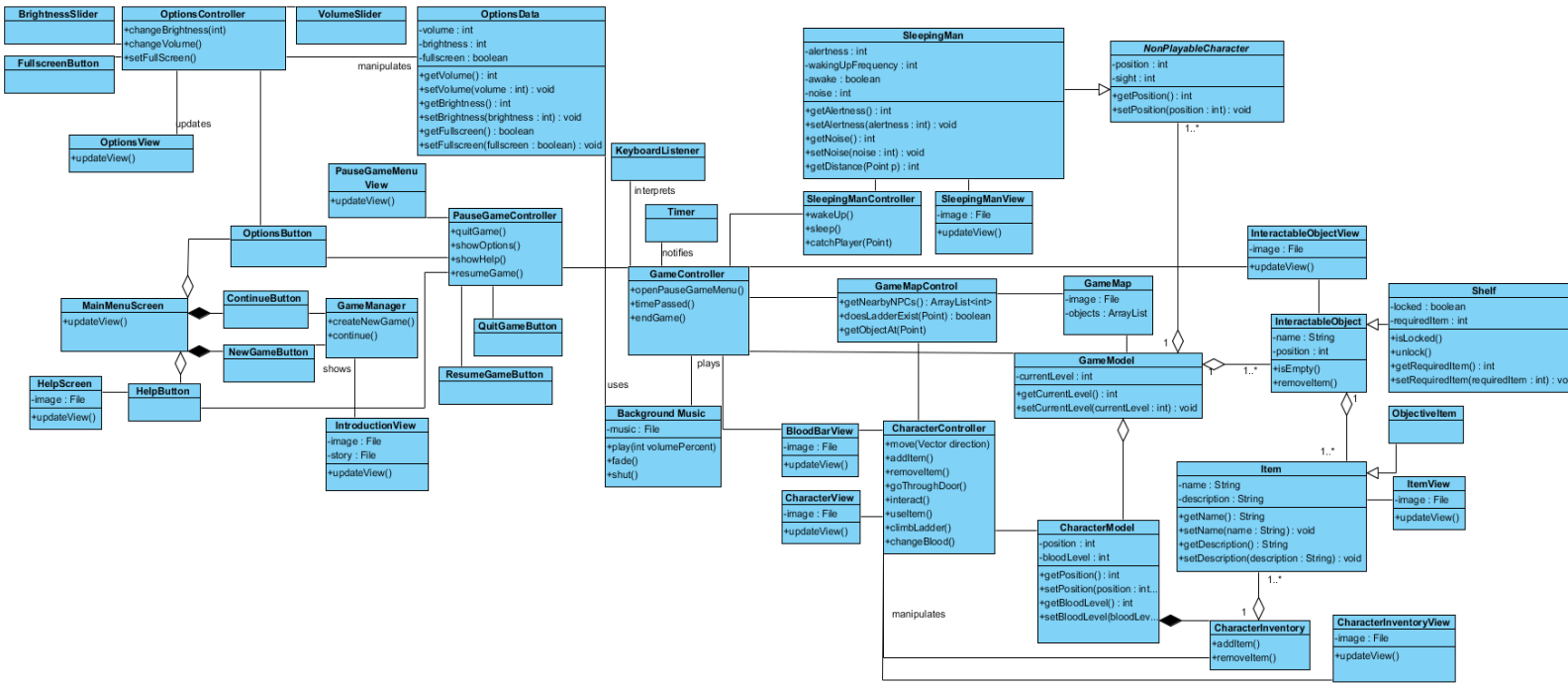


Figure 22 The Complete Class Diagram

This is the complete class diagram of the game. Note that we used a Model View Controller approach, meaning that all entity objects are those classes with -Model postfix or no postfix (such as Item), the boundary objects are the -View classes which the player "sees", or the buttons which the player clicks on. The Control objects have the suffix of Controller/Control. They are generally triggered by buttons or the KeyboardListener which inerprets the inputs of the game and sends the

All Controller classes are responsible for manipulating the data of its model classes and updating the View classes (i.e. repainting them)

Note that although not clear in this diagram, there is a hierarchy between class in creating the instances of them. (i.e. GameController creates Character related classes.) This is explained thorougly in the first Sequence diagram which is 3.2.2.1

Note that although game has levels more then one, there is only one game object and corresponding elements (non-playable characters, items etc.) of the game. The game object and the corresponding game elements are created according to the level of the game from the database. For instance, if the player chooses "New Game", the programme creates the game object which has a GameMap object with the mapFile has been set to first levels map file taken from database, and game only has the first level's items.

The GameManager object is reponsible for doing that, creating GameModel object and all the corresponding elements according to current level. Its association to them is not shown to avoid in the diagram to increase readability.

Note that all Character related movement are handled by the CharacterController, and all the other actions are handled by the GameController. The GameController acts like a master controller and invoke functions of GameMapControl, CharacterController and SleepingManController.

Superclasses for SleepingManController and SleepingManView such as NonPlayableCharacterController could be implemented but since in the initial game we did

not specify any NonPlayableCharacter other than SleepingMan, we did not do this implementation for this time only, to increase readability of the class diagram.

Note that the -Button classes and Timer class are already implemented by Java, hence they are not needed in this class diagram but they are included in it to increase understandibility of the diagram. Also note that minor buttons such as "OK" button or "Back" button are not listed.

Some get/set methods are omitted to increase readability of the diagram.

Unless otherwise is shown, all associations are 1 to 1.

3.2 Dynamic Models

3.2.1 State Chart

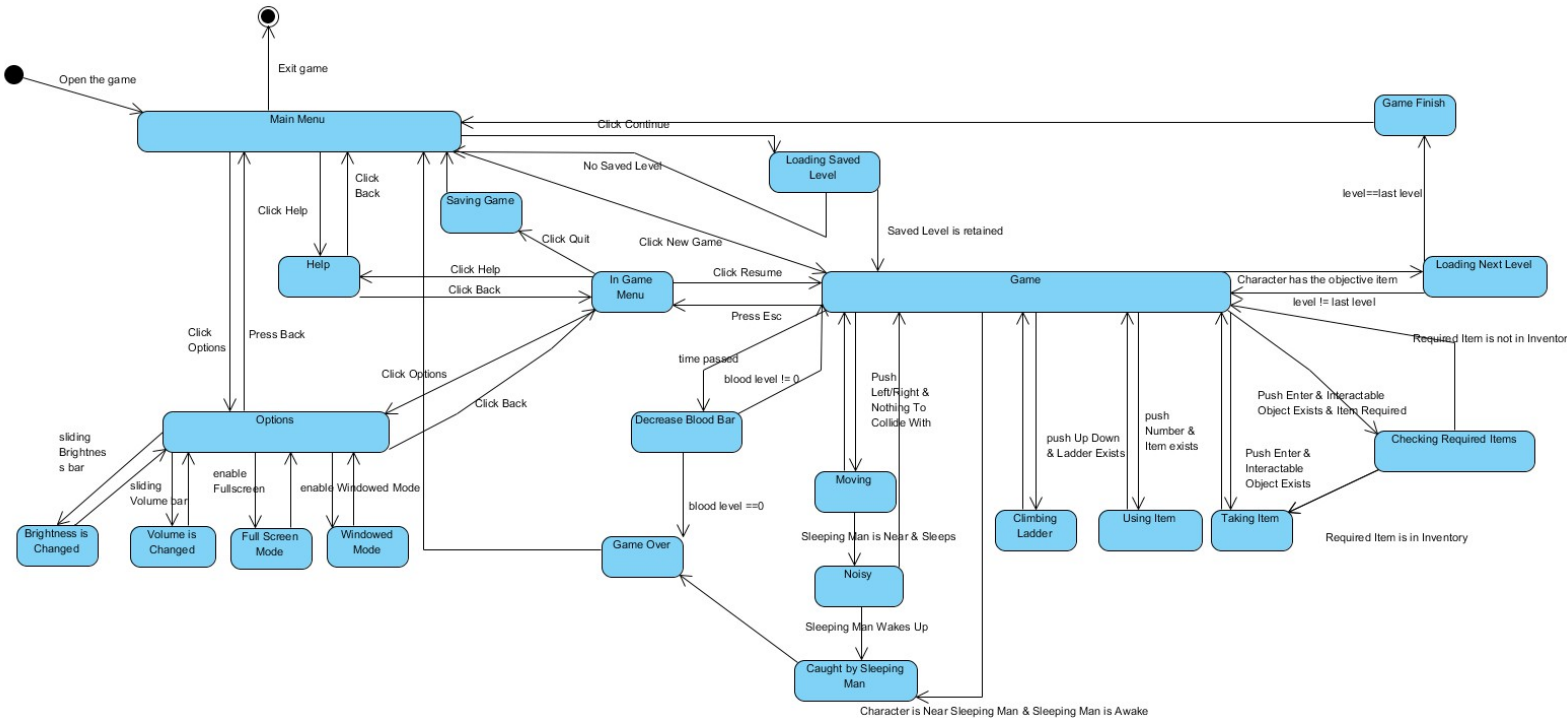


Figure 23 The Complete State Chart

The game starts by the player executing the program. In the Main Menu state, the player can click on buttons to change Options, see Help or to start the game from scratch by clicking the New Game button or Continue button.

When pressed Continue button, the game first changed it state to loading mode, in which the level the player has saved is being loaded. Then it proceeds to the mode, which New Game button immediately proceeds to, the Game mode, which preserves the actual game.

In the Game state, the game changes its states according to inputs from the player. If the player pushes left or right, the game takes the Moving mode which involves the character moving in a direction and updating its view. The Moving may changed to Noisy mode if there is a sleeping man nearby and he sleeps. In Noisy mode, the noise the sleeping man hears is increased and the game checks whether he should wake up or not. If he wakes up, the game's state is changed to "Caught by Sleeping Man" immediately because, logically, the character would already be in the sight of the sleeping man. (This may be changed in the future features of the game.) Then the game is over.

Alternatively, if the sleeping was already awake and near the character while in Game mode, he again catches the player, entering the Caught by Sleeping Man, and then again the game is over. In the case the player pushes Up or Down, the game lets him go up or down as a special case, because there is a ladder where the character is situated in the map, so the game enters Climbing Ladder state.

When the player pushes a number that corresponds an item in the inventory, the game enters Using Item mode, applying the item's benefits and consuming it.

In the case of the player want to interact with an object that contains item behind him, the game first checks if there is such an object then enters Taking Item state, in which the game removes the items from the object and puts into Character inventory. (This state

may be changed in the future if there will be other kinds of interactable objects which are not only for taking the items inside.) In case of the object requires an item to interact (i.e. Key), there is an intermediate step called Checking the Required items for the game.

Note that those actions cannot occur at the same time, i.e. the player cannot use an item while climbing a ladder, so there is no need to use more than one state chart diagram because the game only takes one step at a time.

In case the character has the objective item, the game immediately changes its state Loading Next Level. In this state the game can load next level's game elements and return to Game state again. Or, it can end the game if there is no next level left.

In Options state, which can be reached either by from the In Game Menu which appears when the game is paused or from the Main Menu, the player can slide sliders to change the mode to "Options Data is Changed" mode, i.e. Sliding Brightness bar leads the game state to the brightness is being changed.

3.2.2 Sequence Diagrams

3.2.2.1 Starting A New Game

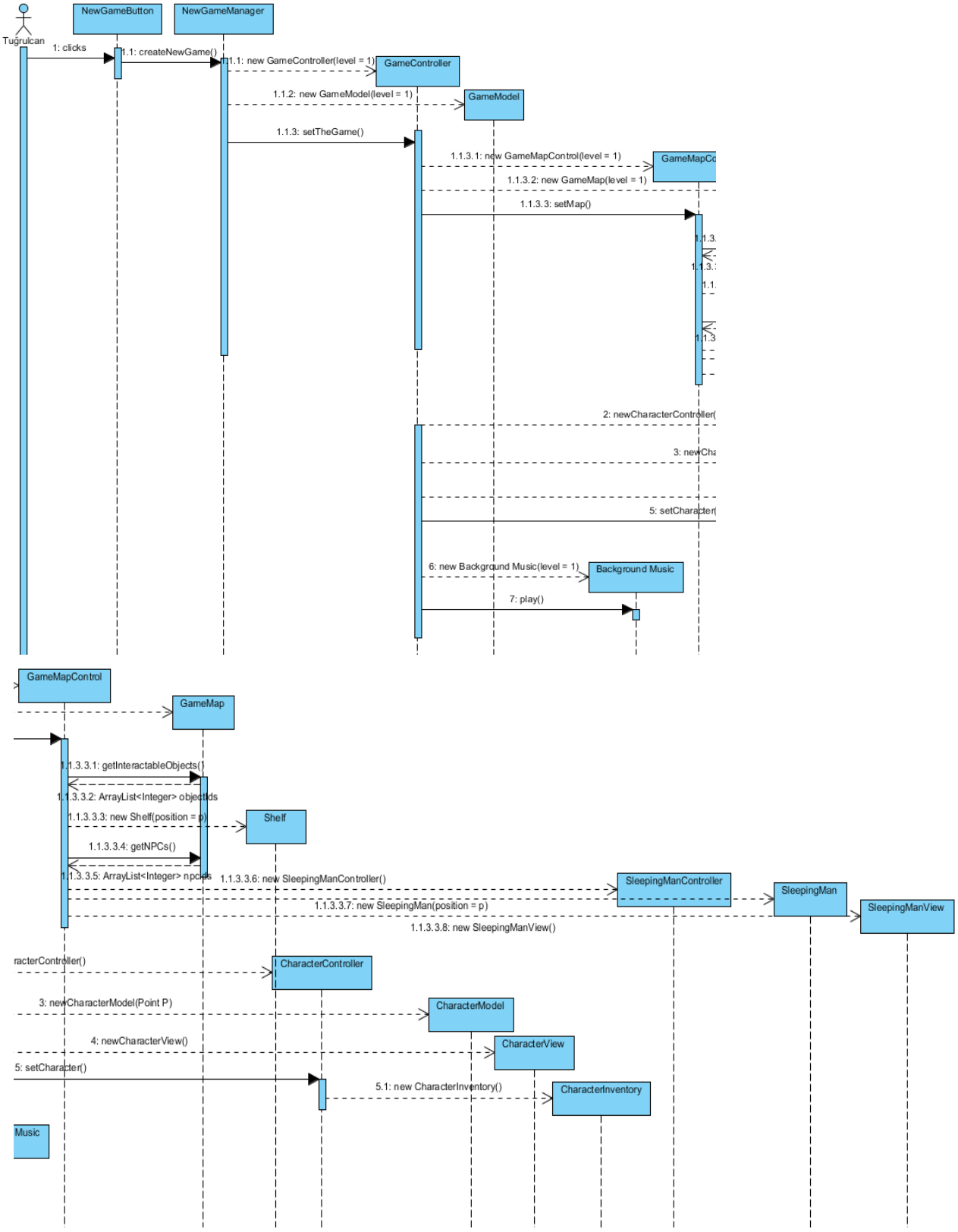


Figure 24 Starting New Game Sequence Diagram

This Sequence Diagram is the initialization steps between Tuğrulcan clicking New Game button from the main menu and Tuğrulcan sees introductory texts and images about the story, which has not been depicted in the diagram (it can be used as a loading screen since the actions in that diagram will take time.) The events are started by Tuğrulcan pressing New Game Button from the main menu. The GameManager, who is responsible for either creating a new game with level = 1 or continue from an old level which is loaded from computer memory, creates "Game" related classes, then every cluster's "Control" class works in a chain of creating other classes. GameMap control gets the list of ID's from the GameMap which stores the information about the map and creates objects and npcs correspondingly. Once the view classes created, the user sees them on the map.

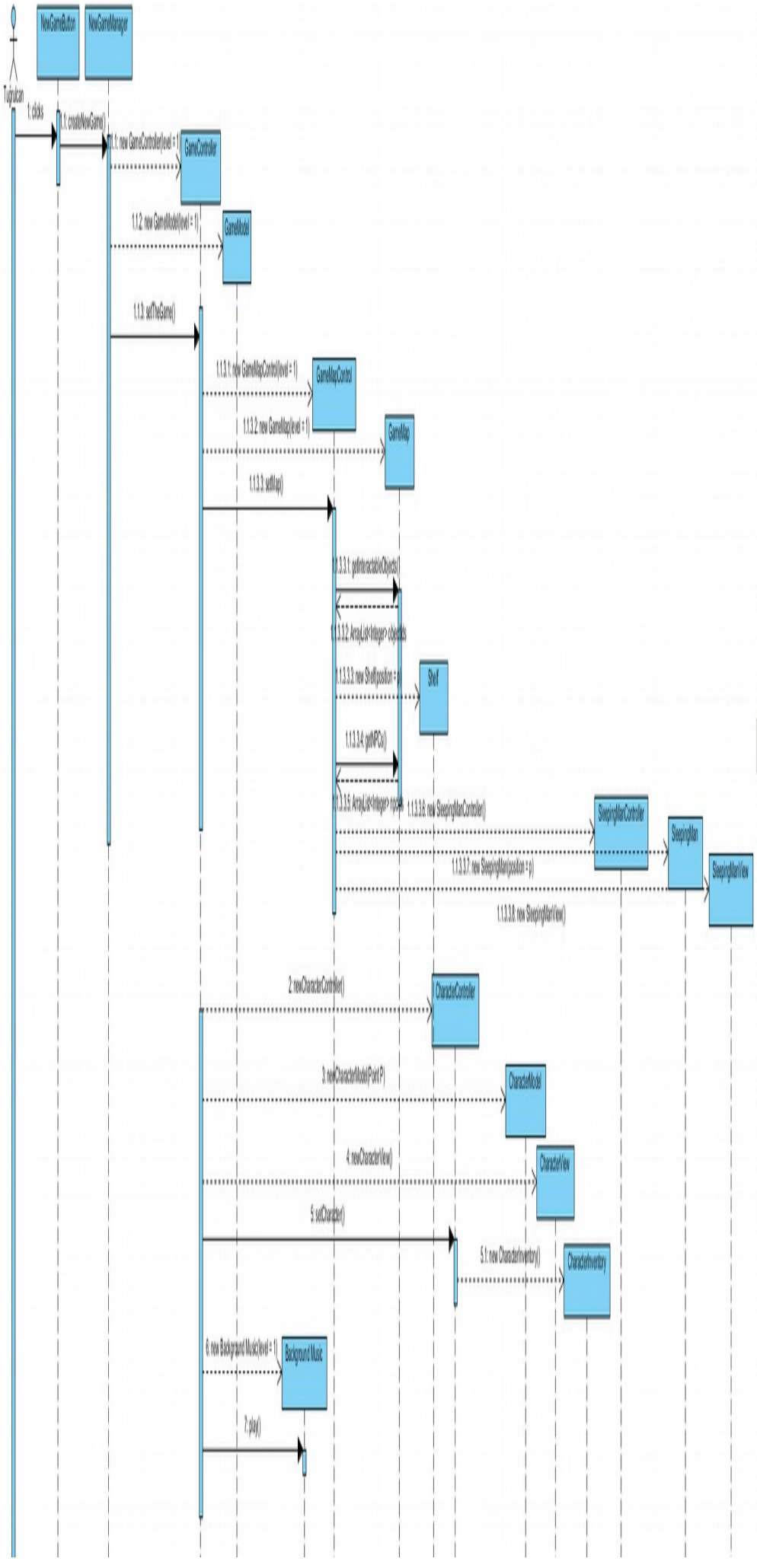
Notes:

1) The objects that are created when the programme has been first started are not listed here. These are the objects that are created during the initilization after starting the game. They are created in this step because they are dependent on the level the game starts. (If GameManager was commanded to continue from another level, it would create different GameModels etc.) Classes such as Timer are created before this step because they are not level dependent.

2) There is a hiearchy between classes so that the control, the view and the model classes for one entity are created by its superclass. As seen in the diagram, GameController creates classes related to GameMap and Character.Also GameMapControl creates classes related to Interactable Objects (shelf in this scenario) and Non-Playable Characters (sleeping man in this case). CharacterController creates classes related to character inventory. The superclasses have access to its subclasses.

3) Although SleepingManController, SleepingMan and SleepingManView are created earlier than Character and its related classes, they are shown in their right to make the diagram more readable.

4) GameMap is static, meaning that there is nothing going on on the map, it is just a background. So it does not update itself thus, it is a model class with a piece of view (a mere image) with no update. There is no need to have a view class for the maps. The objects' views are updated instead.


NewGameButton
NewGameManager
GameController
GameModel
GameMapControl
GameMap
Shelf
SleepingBagController
SleepingBag
SleepingBagView
CharacterController
CharacterModel
CharacterView
CharacterInventory
BackgroundMusic
1: clicks
1.1: createNewGame()
1.1.1: new GameController(level = 1)
1.1.2: new GameModel(level = 1)
1.1.3: setTheGame()
1.1.3.1: new GameMapControl(level = 1)
1.1.3.2: new GameMap(level = 1)
1.1.3.3: setMap()
1.1.3.3.1: getInteractableObjects()
1.1.3.3.2: ArrayList<Integer> objectIDs
1.1.3.3.3: new ShelfPosition = n)
1.1.3.3.4: getNPCs()
1.1.3.3.5: ArrayList<Integer> npcIDs
1.1.3.3.6: new SleepingBagController()
1.1.3.3.7: new SleepingBag(position = n)
1.1.3.3.8: new SleepingBagView()
2: newCharacterController()
3: newCharacterModel(PlayerID)
4: newCharacterView()
5: setCharacter()
5.1: new CharacterInventory()
6: new BackgroundMusic(level = 1)
7: play()

3.2.2.2 Climbing A Ladder

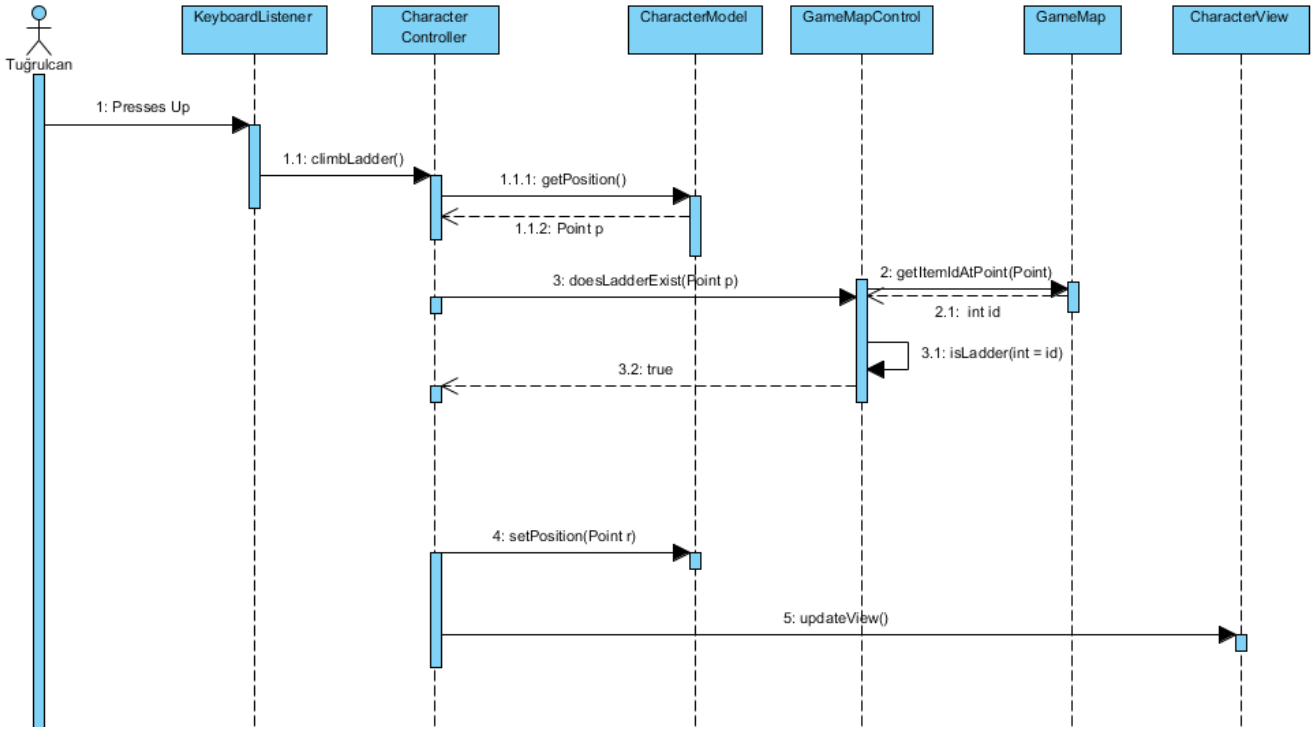


Figure 25 Climbing A Ladder Sequence Diagram

Once got his character in front of a ladder, Tuğrulcan presses Up to make him climb up to ladder. CharacterController's, which is responsible for character-related movement, function climbLadder() is invoked. CharacterController gets character's point in coordinate system and checks if there is a ladder in that point. GameMapControl gets the id of the item in this point from the GameMap and checks if the id corresponds to a ladder. Then it returns true so that CharacterController can move the Character by setting its new position.

Note that ladder is a long thing that takes up a large space in the vertical axis. Tuğrulcan has to press Up for sometime so that this process will repeated until the character climbs the entire ladder and reaches the upper floor.

3.2.2.3 Taking an Item From The Shelf

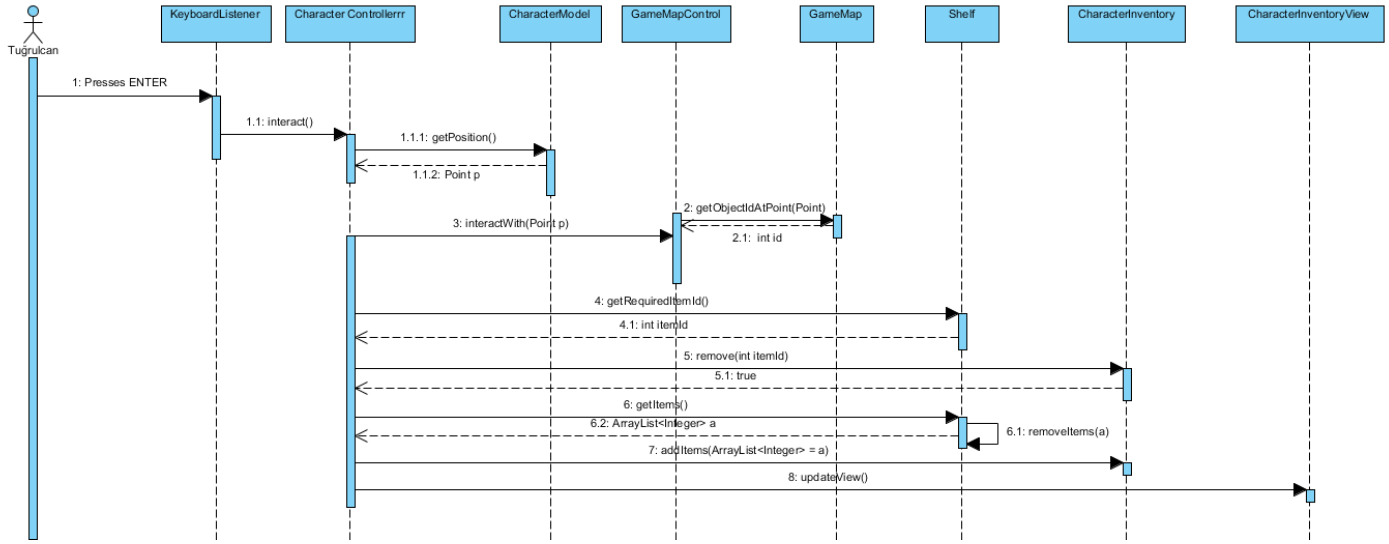


Figure 26 Taking an Item From The Shelf Sequence Diagram

Once in front of a shelf, Tuğrulcan hits Enter to interact with it. Character controller first gets the character's current position and get which interactable object is situated in that point by calling GameMapControl's interactWith(Point) function. Then it calls the corresponding object's getRequirementItemId() to learn what this objects requires to interact with. The Shelf is locked and requires key. CharacterController removes key from character's inventory and adds the items in the shelf, removing the corresponding items from the shelf and updating the inventory shown in screen.

Note that if the character is not in front of any item, the sequence is over before getRequiredItemId() because there will be no Shelf to call its function.

Also note that if getRequiredItemId() returns -1, CharacterController won't call CharacterInventory's removeItem() method since it is not required.

3.2.2.4 Sleeping Man Wakes Up And Catches The Player

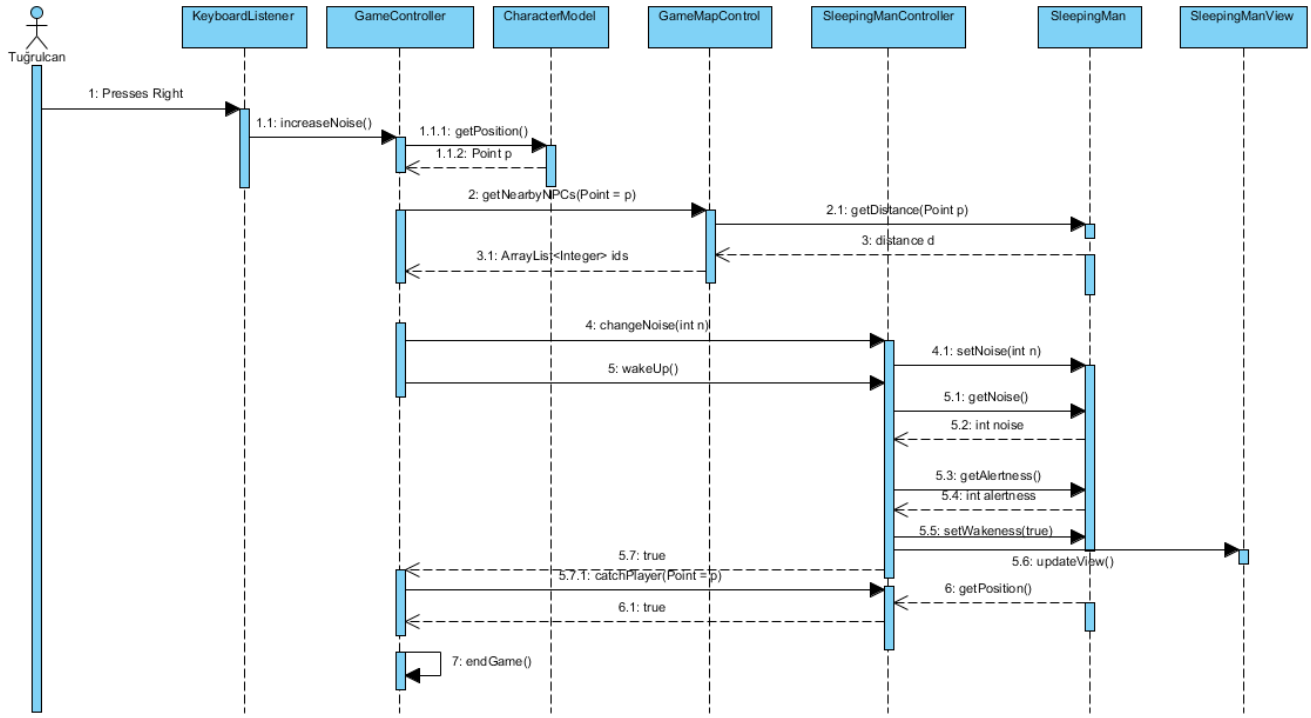


Figure 27 Sleeping Man Wakes Up And Catches The Player Sequence Diagram

Tuğrulcan is close to the sleeping man who sleeps and moves around him. Every time he moves (in this case, presses right), the GameController gets the character's position and find the nearby Non-Playable Characters, in that case, nearby Sleeping Man's. With some calculation that involves the character's distance to the sleeping man, it increases the noise that sleeping man hears. It then checks if he is going to wake up by invoking SleepingManController's wakeUp() function. This functions uses some calculations implemented in itself that involves the noise the man hears and its alertness, then wakes him up accordingly (and updates its view.) Since the wakeUp() function has returned true, the GameController also checks if the Sleeping Man can catch the character by comparing their distance to each other. If that is the case, the GameController ends the game.

3.2.2.5 BloodBar Goes Empty

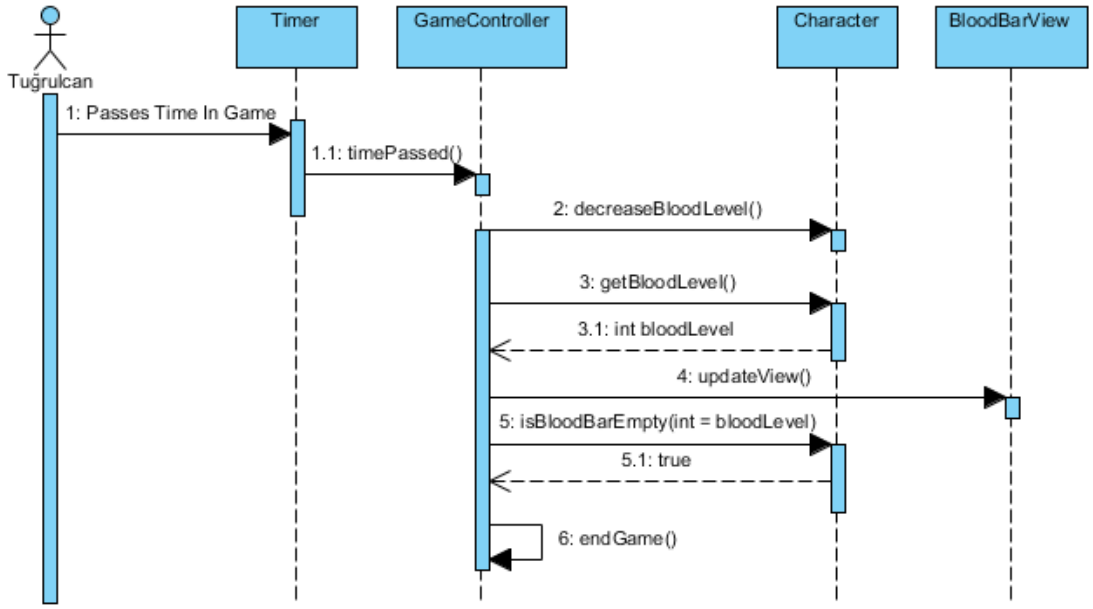


Figure 28 BloodBar Goes Empty Sequence Diagram

As Tuğrulcan passes time in game without pausing it, the Timer notifies GameController a unit of time has passed with a preset frequency. Then GameController decreases the blood level of the character, updates the blood bar in the screen and gets the new blood bar in order to check if it is empty. Since in this scenario, Tuğrulcan has passed too much time without refilling the Blood Bar, the character's blood level is zero, so the game is over and GameController

Note that the blood level should be an attribute of the character because CharacterController may make the character use an item to refill it.

3.2.2.6 Changing Brightness

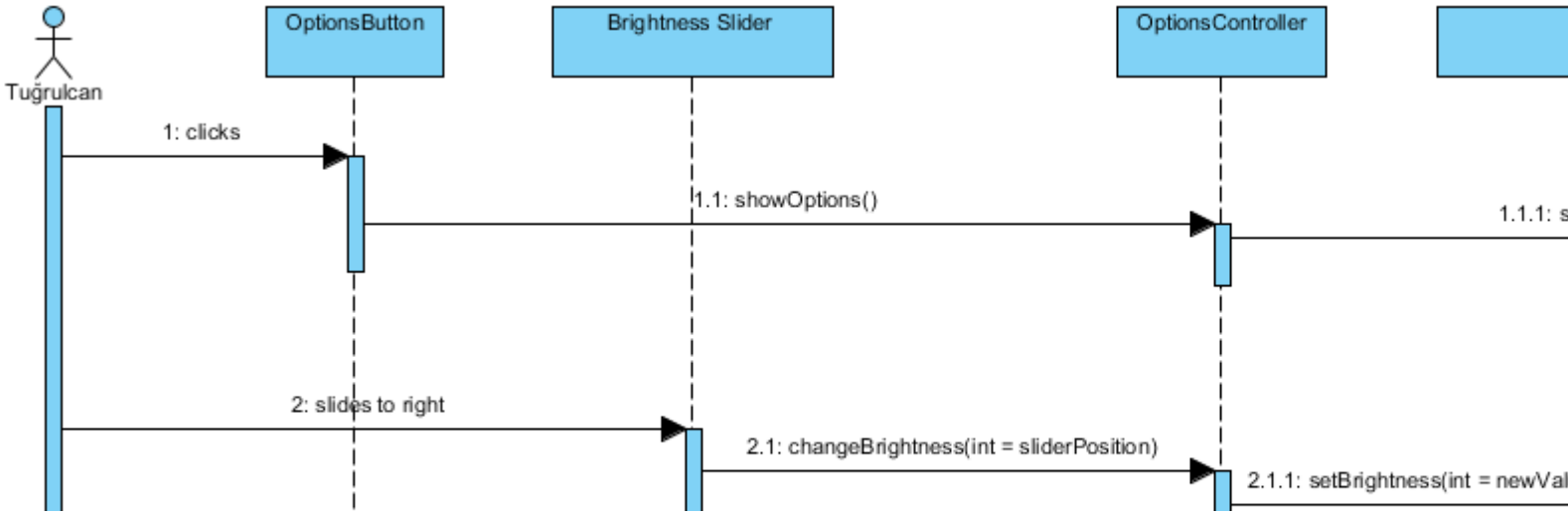


Figure 29 Changing Brightness Sequence Diagram

Either from the main menu or from the in-game menu (does not matter), Tuğrulcan clicks on "Options" button to change the brightness. OptionsButton urges OptionsController to paint the OptionsView in the front. Then Tuğrulcan slides the Brightness slider to right. Brightness slider sends the new position of the slider to the OptionsController then according to this new position, OptionsController generates and sets the new value of the of brightness in OptionsData and updates the OptionsView so that Tuğrulcan will actually see slider's new position.

Note that we have not decide on the implementation details now, so the the problem "How the actual brightness of the screen changes?" is out of question right now.

4 Conclusion

In this report, we have explained our game. We started first with the motivation behind this game's idea. Then, we explained the game itself, its objectives and its features. The gameplay is explained by use cases and scenarios. The matter what should we have as object for those use cases is explained in the class diagram. The model view controller approach is used in the class diagram. How are those approach is implemented in scnearios ,in terms of the algorithm, is explained in the sequence diagram. The state chart acts as a simple overview of sequence diagrams. They are all made by the program Visual Paradigm. We hope to realize this ideas in a concrete game in the future.