Bilkent University

Department of Computer Engineering

Bilkent University

CS 319

Object-Oriented Software Engineering

Fall 2014

Dungeon Escape

Final REPORT

December 25th, 2014

GROUP #10

Ahmet Emre Danışman

Ateş Balcı

Buket Depren

Batuhan Berk Yaşar

Ayşe İrem Güner

Contents

1 Introduction

Object-oriented software engineering is a relatively new concept. It is a way of modeling real structured systems into software based applications. Even though many people build up the idea that software engineers are just a bunch of coders, it is completely the other way. The design stage is a whole process by itself and it is probably the most intricate one in completing software project.

As our group project, we decided to develop a 2D side scrolling-puzzle platformer game called ‘Dungeon Escape’. There are many side scrolling and puzzle platformer games in the market with variety of user interfaces and features. Some of these games, such as Super Mario Bros, Donkey Kong and Little Big Planet became legends and are the most successful games of this kind. The aim of these games is to go to a finish point from a starting point in a map while you are dealing with numerous puzzles and game mechanic testers. The number and strength of enemies (if there are any), the tools that players use and difficulty of puzzles vary from game to game depending on games’ topic and developers’ decision.

In the Dungeon Escape, player will try to advance through levels by solving built-in puzzles via usable tools (boomerang and rope), gates, platforms, triggers, buttons and levers. Interaction with map and how to use those tools will be explained in the ‘Help’ menu which is accessible from the main menu to make players get familiar with the game quickly. Since difficult games get a lot attention nowadays, most of the game’s levels will be challenging enough to preserve player’s interest and attention. The real aim of the game is to challenge player with its difficult puzzles and mechanics.

Dungeon Escape provides the player both default levels and map editor in order to create their own map and choose own tools using in the game. Thus, the players can play their own levels easily.

2 Case Description

The main aim of the side scrolling and puzzle platformer games are challenging players by providing different levels of puzzles. Also a purpose of any game is entertaining people with enjoyable scenarios. Dungeon Escape has 9 different default levels that our main character will try to go through and the player can choose any of them or use the level editor to create new levels in case the player does not want to play default levels.

Side scrolling genre is the one of the first genres that appeared in the history of gaming. If you think about all successful games of this genre, you realize one common point: difficulty. Most of the games are so challenging that even their first level is harder than a nowadays' ordinary game's final boss. So, in order to make Dungeon Escape a good side-scrolling puzzle platformer, default levels will be designed carefully. First levels will be introductive, will teach player how to use tools and how to interact with the map. As users keep playing, more challenging levels will greet them. These levels may be too much for regular people, but these are the only ones that can satisfy hardcore gamers.

3 Requirement Analysis

3.1 Overview

The project has been developed is Dungeon Escape as a type of strategy game. Dungeon Escape supports creative and analytical thinking. In Dungeon Escape, the main character will try to escape from a sci-fi themed prison filled with puzzles by using some items and tools. Puzzles will be solved by pulling some levers, pushing some buttons, using rope to escalate and jump over platforms, using boomerang to trigger some movable objects and take advantage of platforms. These levers show that which triggered object will be activated by pulling that lever with a green line. As you reach the finishing point of a level, you will advance to the next level.

There are 9 levels that include different platforms that the main character will go through and an editor mode that provides creating completely different levels according to the user’s desire in Dungeon Escape. Design and atmosphere of the levels will be related to the story. In the following levels, our char uses either rope or boomerang to reach up to finishing point. If you die on your way to the finishing point, you will restart that level. There are also check points in this game. If you die after passing these points, you will restart that level from that point. The main character will be controlled by both the keyboard and the mouse. The user will move the character, select tools and items to use and interact with objects on the map via the keyboard. Mouse will be used to use tools, shift the camera view and use the menu. Dungeon Escape also lets the user to alter the keys of the game by using menu options.

3.2 Functional Requirements

- Play: Opens a screen with a list of 9 premade levels and custom created levels to choose from. Custom levels projects the levels created by the user.
  -User should be able to select one of default levels or custom level then start the game.

- Editor: opens a screen with list of tools that can be used to create a new level. User is able to create a level with respect to user's wish. While creating the level, user starts over from scratch.
  -User should be able to put obstacles with different height and width and determine the location.

  -User should be able to put moveable object with fixed size and determine the location.

  -User should be able to put door object with fixed size and determine the location.

  -User should be able to put platform with different height and width, decide whether or not this obstacle will travel vertically or horizontally, travel range of it and determine the location.

  -User should be able to put contactTrigger which is an area and triggered by both moveableObject, items that are used by the main character and the main character himself, determine its delay time and the location.

  -User should be able to put lever which is a guide way and triggered by wıth moveableObject and the user, determine its triggered duration and the location.

-User should be able to put playerTrigger which is an area and triggered by only the char, determine its delay time and the location as well as its type (checkpoint, level end, contactTrigger).

-User should be able to remove any type of element.

-User should be able to change spawn coordinates by removing mouse.

-User should be able to alter the background image by using backgroundUrl box (ex: back.png)

-User should be able to save and load the created level with a preferred name.

-User should be able to place a welcoming image for each level called 'Tip'.

- Options: Opens a screen with a list of button control mechanism. These buttons are controls the character's actions.

  -User should be able to change the controlling keys for jump function, go left function, go right function, interact object function, restart function.

  -User should be able to back to the main menu after the alterations made in controlling keys

- Exit:
  -User should be able to exit the game

3.3 Non-Functional Requirements

3.3.1 Performance Requirements

- Game should be running at a high and stable frame rate in order to give the user a better real time platformer experience. 60 FPS is a good target frame rate in order to provide this experience.
- Map editor mode must be easy to practice for the user.
- Inexperienced players should be able to start a new game within 60 seconds.
- Levels are created from easy to difficult to warm users to the game.

3.3.2 Accuracy Requirements

- Although physics algorithms are not applied exactly, physics algorithm should still be moderately accurate in order to give the user a better experience in Dungeon Escape game. A more accurate physics algorithm would definitely increase the player's satisfaction during gameplay.

3.3.3 Ease of Learn Requirement

- The user is able to play the game easily because the user is already familiar with the tools (rope and boomer) in the game and these tools are rolled according to the physics algorithm.

3.3.4 Implementation Requirements

- Modifying the game should not be a difficult task for players that are interested in. By using object-oriented programming, this feature is available

3.4 Constraints

The implementation language of Dungeon Escape game should be Java because java is comparatively useful for programming language for Object Oriented Software Design. Java is an efficient language because it has many control mechanisms and it supports many libraries. Java is used in this project because our project needs to meet the needs of the Object-Oriented Software Engineering course.

3.5 Scenarios

Scenario Name: Restart the game

Participating Actor Instances: Buse: Player

Flow of Events:

1. Buse selects “Play” button from Main Menu
2. Buse selects level1.
3. Because 1st level's weapon is boomer, Buse play the game with boomer.

4. Buse goes through first check point.
5. Buse needs to use moveable object to jump higher.
6. Buse tries to change place of the moveableObject with boomer.
7. Moveable object falls infinitely down.
8. Buse can’t finish the level.
9. Buse restarts the game with the key “R”.
10. Buse completes the level.

Scenario Name: Start the game from the current check point.

Participating Actor Instances: Buse: Player

Flow of Events:

1. Buse selects “Play” button from Main Menu
2. Buse selects level2.
3. Because 2nd level’s weapon is rope, Buse play the game with rope.
4. Before Buse goes through first check point, Buse falls to infinity.
5. Buse start the level from the beginning.
6. Buse uses rope as a weapon and climb over the vertical wall.
7. Buse can finish the level.
8. Level ended.

Scenario Name: Map editor

Participating Actor Instances: Buse: Player

Flow of Events:

1. Buse selects “Editor” button from Main Menu.
2. Buse starts to create a new level.
3. Buse selects “Moveableobject” button from the menu in Editor option
4. Buse clicks on the moveableObject from the **left upper corner** then relocates it as Buse desires.
5. Buse clicks on “PlayerTrigger” button from the menu in Editor option
6. Buse sets 1 as a triggerDuration.
7. Buse clicks on playerTrigger and puts it near the moveableObject.
8. Buse clicks on “Door” button from the menu in Editor option
9. Buse clicks on it and puts somewhere 300 pics far away the PlayerTrigger.
10. Buse finds out that she is not able to go through the door in that triggerDuration.
11. Buse decides to put a contactTrigger instead of playerTrigger.

12. Buse clicks on playerTrigger than clicks on “Remove Element” button then clicks on contactTrigger and relocate it where the playerTrigger was.
13. Buse clicks on “Platform” button from the menu in Editor option
14. Buse resizes the width and height of platform then locates it.
15. Buse decides that level is good enough to play.
16. Buse decides the weapon will be used in the level.
17. Buse give this custom level a specific name (customLevel1)
18. Buse saves this level by clicking on “Save” button.
19. Buse loads this level by clicking on “Load” button.
20. Buse use “Esc” key to exit.
21. Buse goes back to the Main Menu.

Scenario Name: play custom level

Participating Actor Instances: Buse: Player

Flow of Events:

1. Buse selects “Play” button from Main Menu
2. Buse selects customLevel1.level
3. Buse play the game with the weapon she decided in Editor mode.
4. Buse goes through all check points.
5. Buse completes the level.

Scenario Name: using Option Menu.

Participating Actor Instances: Buse: Player

Flow of Events:

1. Buse selects “Play” button from Main Menu
2. Buse selects level2.
3. Buse is not comfortable with the direction keys during the game.
4. Buse presses “Esc” key and then select “Exit” button.
5. Buse goes back to the Main Menu.
6. Buse selects “Option” button from Main Menu
7. Buse selects “Jump Button: W” button (W was the key for jump function).
8. Buse sees “Press any key to set” button.
9. Buse press “shift” “key.

10. “shift” key assigned to jump function (“Jump Button: shift”)
11. Buse decides not to change other keys.
12. Buse selects “Back” button.
13. Buse goes back to the Main Menu.
14. Buse selects “Play” button from Main Menu
15. Buse selects level2.
16. Buse starts the level2 from the beginning.
17. Buse uses rope as a weapon and climb over the vertical wall.
18. Buse can finish the level.

Scenario Name: Resume/Exit option in Dungeon Escape

Participating Actor Instances: Buse: Player

Flow of Events:

1. Buse selects “Play” button from Main Menu
2. Buse selects level1.
3. After Buse plays a while, phone rings.
4. Buse press “Esc” key to stop the game.
5. When Buse come back, she selects the “Resume” button.
6. Buse continues the game.
7. Buse decides to play Dungeon Escape after she goes to gym.
8. Buse press “Esc” key to stop the game.
9. Buse selects the “Exit” button.
10. Buse goes back to the Main Menu.
11. Buse selects the “Exit” button.
12. Buse closes the game.

Scenario Name: using Credits Menu.

Participating Actor Instances: Buse: Player

Flow of Events:

1. Buse is curious about who created Dungeon Escape game.
2. Buse selects “Credit” button from the Main Menu.
3. Buse learns the creators.
4. Buse selects “Back” button.

5. Buse goes back to the Main Menu.

3.6 Use Case Models

3.6.1 Textual Use Case

Use Case Name: Play Game

Primary Actor: Player

Stakeholders and Interests:

-Player aims to complete all levels.

Pre-condition: At the beginning the game settings are set as default.

Entry Condition: Player selects “Play Game” button from Main Menu.

Exit Condition: Player presses “Esc” key then Pause Menu or closes the game window.

Success Scenario Event Flow

1. Game is started after the Play Game option has been chosen.

2. Player starts form the first level.3

3. Player plays until the finish point of the first level is reached.

Use Case Name: Options

Primary Actor: Player

Stakeholders and Interests:

-User wants to change game settings: changing the background, changing the sound options etc.

-The game will be updated once the user is done.

Pre-condition: The game settings are set as default.

Post-condition: Game settings are updated.

Entry Condition: User choses “Options” from main menu.

Exit Condition: User choses “Back” option to return to main menu.

Success Scenario Event Flow:

1. User choses the “Options” option from the main menu.

2. Game settings will be displayed to user on the screen and user is able to change the key preference.

3. User may or may not change the displayed settings.

4. Any changes will be applied to the game when the user is done.

Use Case Name: Editor

Primary Actor: Player

Stakeholders and Interests:

-Player wants to create a new level according to the user’s desires.

Pre-conditions: Player should be in Main Menu.

Entry Condition: User choses “Credits” from main menu

Exit Condition: Player presses “Esc” key to return main menu.

Success Scenario Event Flow:

1. User clicks the Editor button in the main menu.

2. System displays a new blank level.

3. User creates a completely different level by the tools.

4. User saves that level with a specific name by using “Save” button.

5. User loads that level in custom level by using “Load” button.

Use Case Name: Options

Primary Actor: Player

Stakeholders and Interests:

-User wants to change game settings: changing the background, changing the sound options etc.

-The game will be updated once the user is done.

Pre-condition: The game settings are set as default.

Post-condition: Game settings are updated.

Entry Condition: User choses “Options” from main menu.

Exit Condition: User choses “Back” option to return to main menu.

Success Scenario Event Flow:

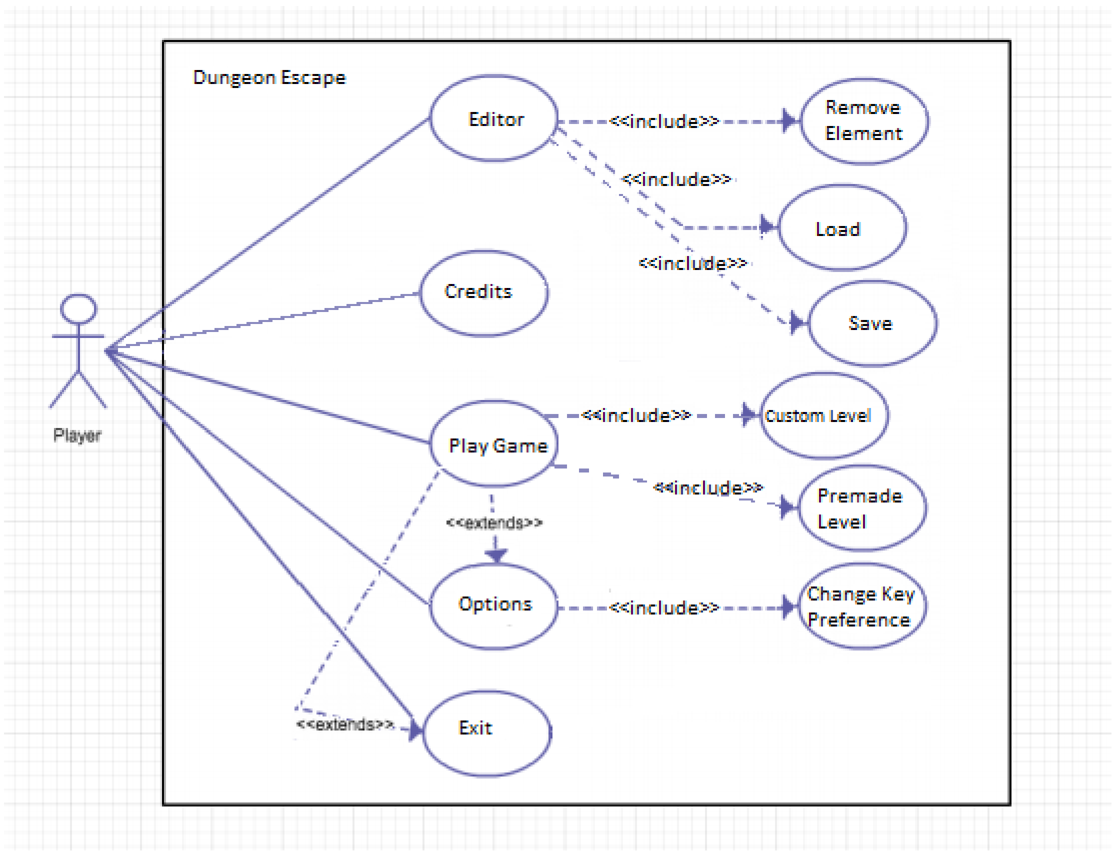
1. User choses the “Options” option from the main menu.

2. Game settings will be displayed to user on the screen and user is able to change the key preference.

3. User may or may not change the displayed settings.

4. Any changes will be applied to the game when the user is done.

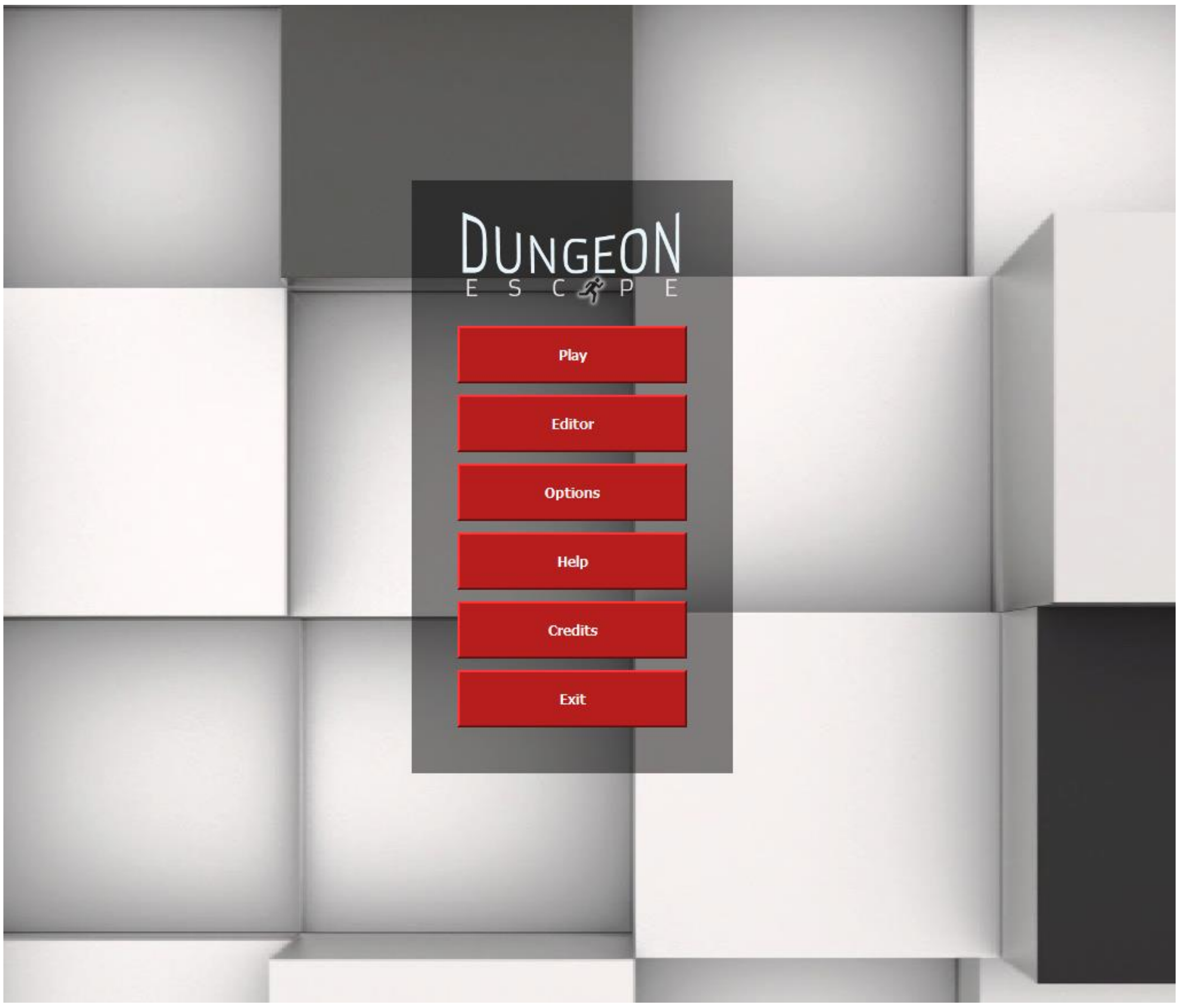
3.6.2 Visual Use Case



Use case diagram of the Dungeon Escape game system is above. Our system basically consists of 5 use cases. Play game, editor, credits, options and exit. Dungeon Escape use case is the one for the user to cover the concept of the game. User is able to select either the custom level or premade levels in play game mode. User is able to save and load created level and remove element from that level in editor mode. User is able to change the key preferences in option mode. User is able to view the creators of Dungeon Escape in credits mode. User is able to exit the game any time with exit mode. In the following of the report, these use cases will be explained further by examining them.

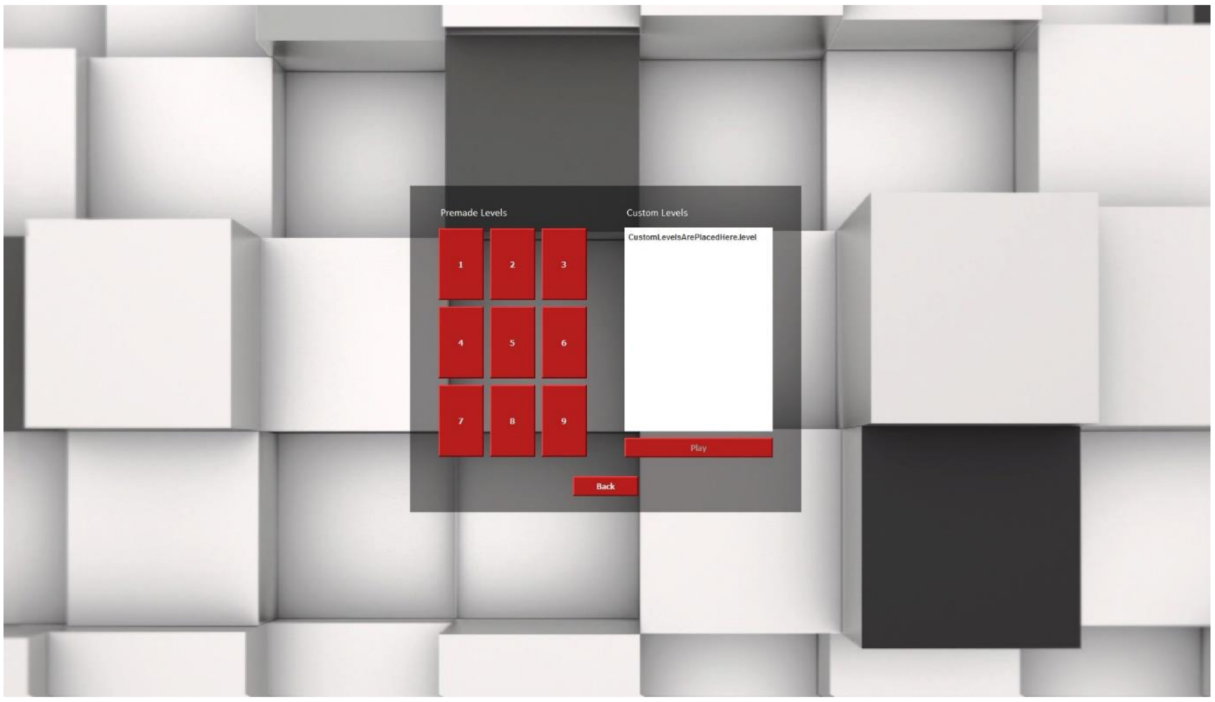
3.7 User Interface/Screen Mockups

3.7.1 Main Menu



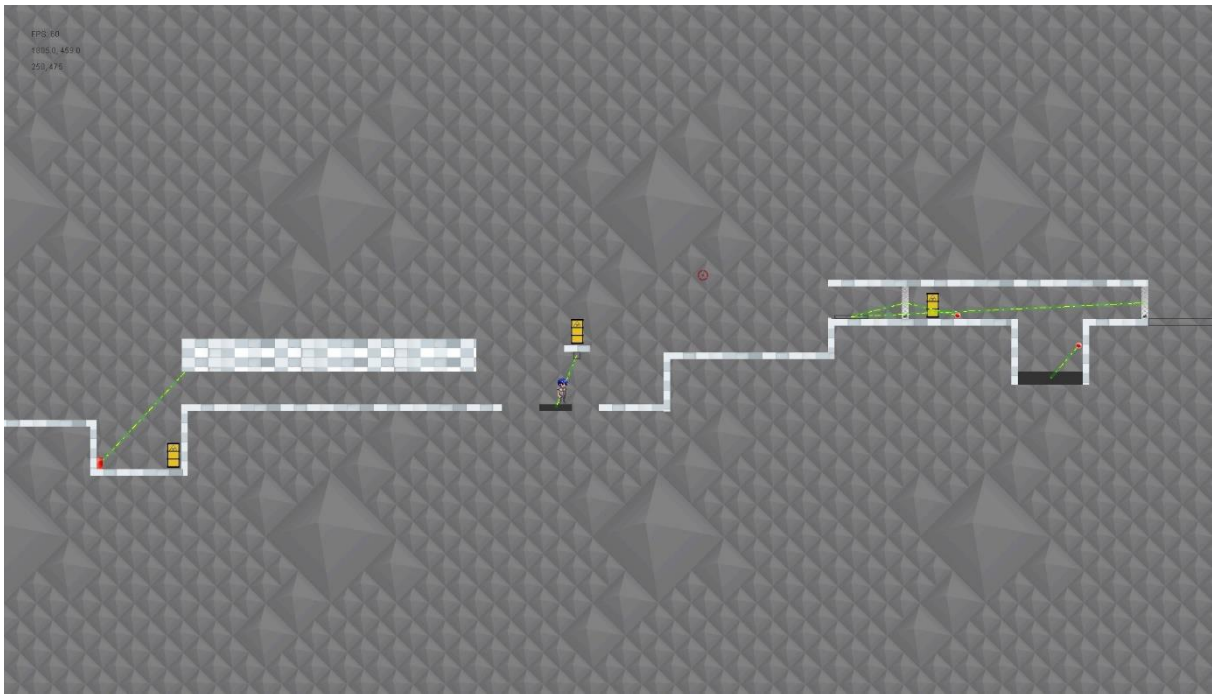
This is the screen that welcomes user when Dungeon Escape is started. Provided selections are ‘Play’ to select a premade or custom level and play the game, ‘Editor’ to create a new level or modify an existing one, ‘Options’ to change key bindings, ‘Credits’ to see developers, ‘Help’ to get additional info about the game and ‘Exit’ to leave the game.

3.7.2 Play Screen



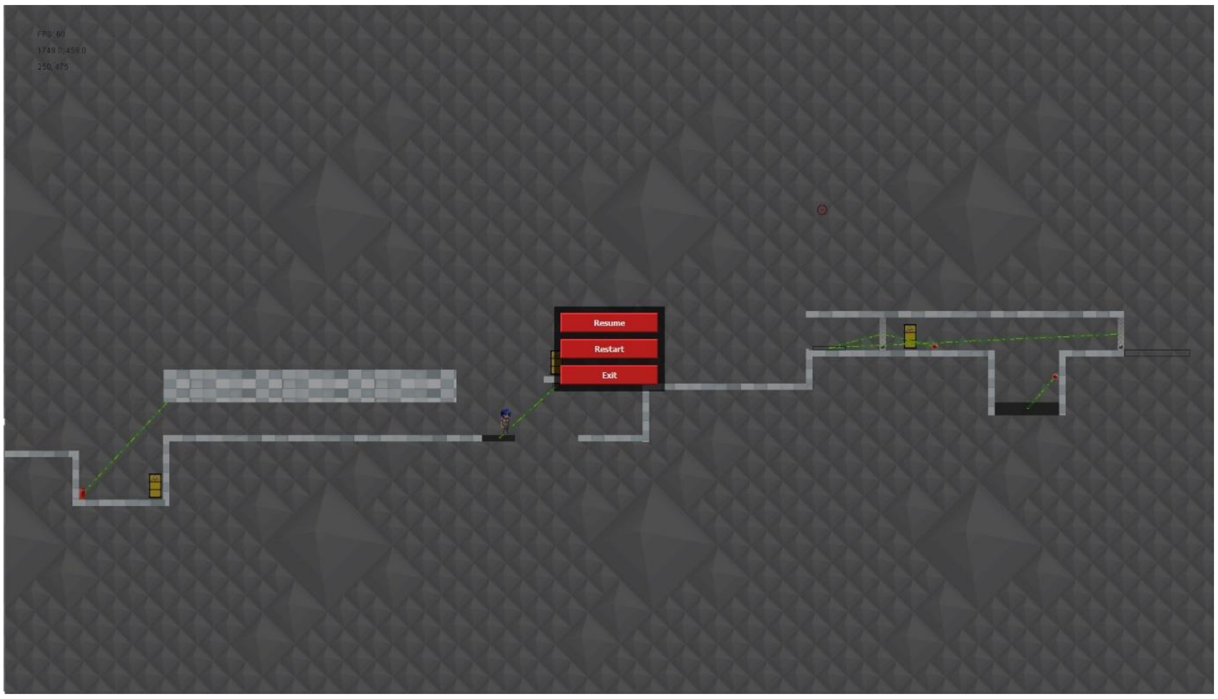
In this screen, user can select one of 9 premade levels by clicking on the numpad or select a custom level from the list.

3.7.3 Gameplay Screen



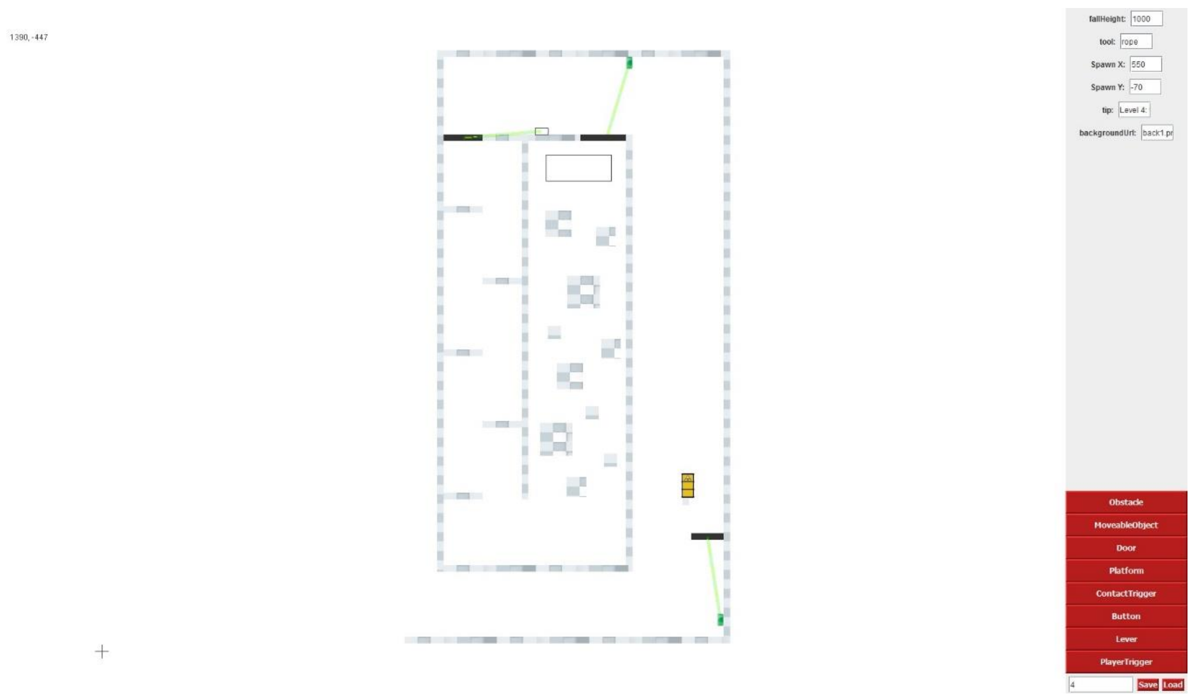
This is the screen that user plays the game. There are not any selections in this screen but they can be reached by the ‘pause menu’.

3.7.4 Pause Screen



User can return to game by clicking ‘Resume’, restart the level or go to the play menu.

3.7.5 Editor Screen


1390, -447

fallHeight: 1000
tool: rope
Spawn X: 550
Spawn Y: -70
tip: Level 4:
backgroundUrl: back1.pr

4

In this screen, user can modify an existing level or create a new one by using all the tools available in the game. As well as shape of the map; background image, welcoming message, spawn coordinates, tool that Is used and fallHeight can be selected too.

Obstacle: Fixed on map, non-intractable, various sized objects.

MovableObject: Fixed sized, non- intractable objects that are effected from PhysicsEngine (momentum, gravity).

Door: Fixed sized, triggerable objects.

Platforms: Various sized, triggerable objects.

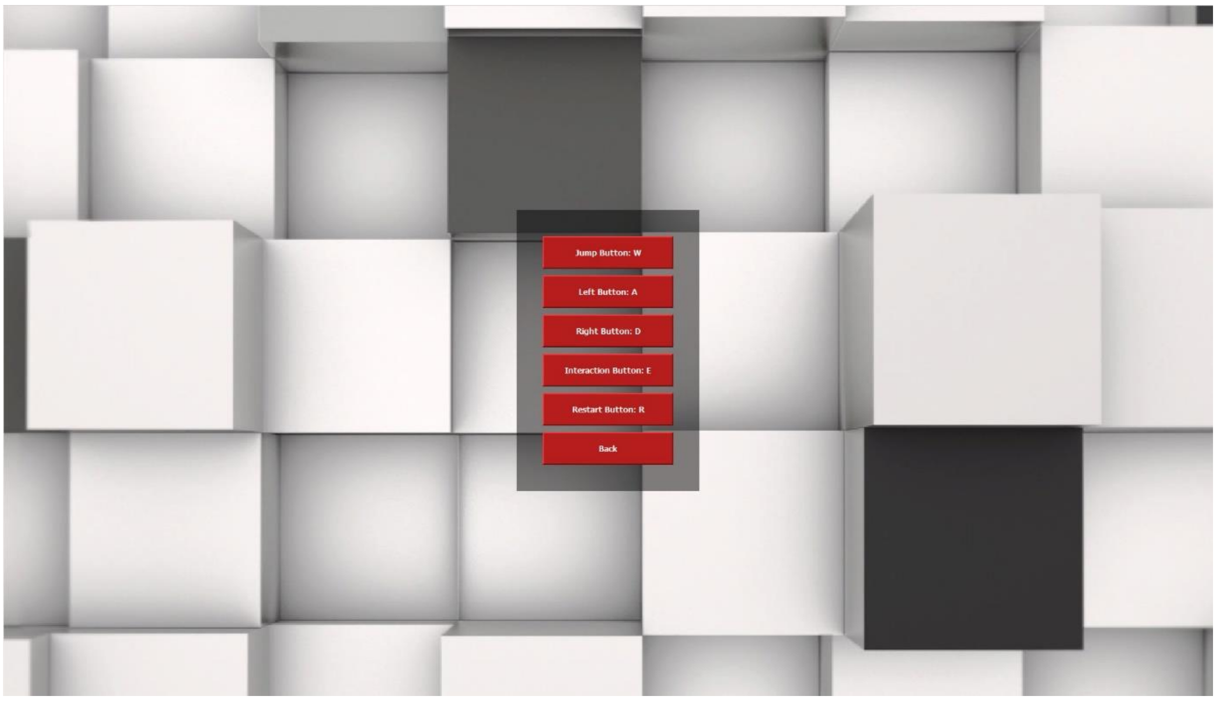
ContactTrigger: Various sized, intractable objects that are interacted by any object that enters its area (main character, tools, movebalObject).

Button: Fixed sized, intractable objects that triggers for a selected amount of time.

Lever: Fixed sized, intractable objects that triggers in toggle system.

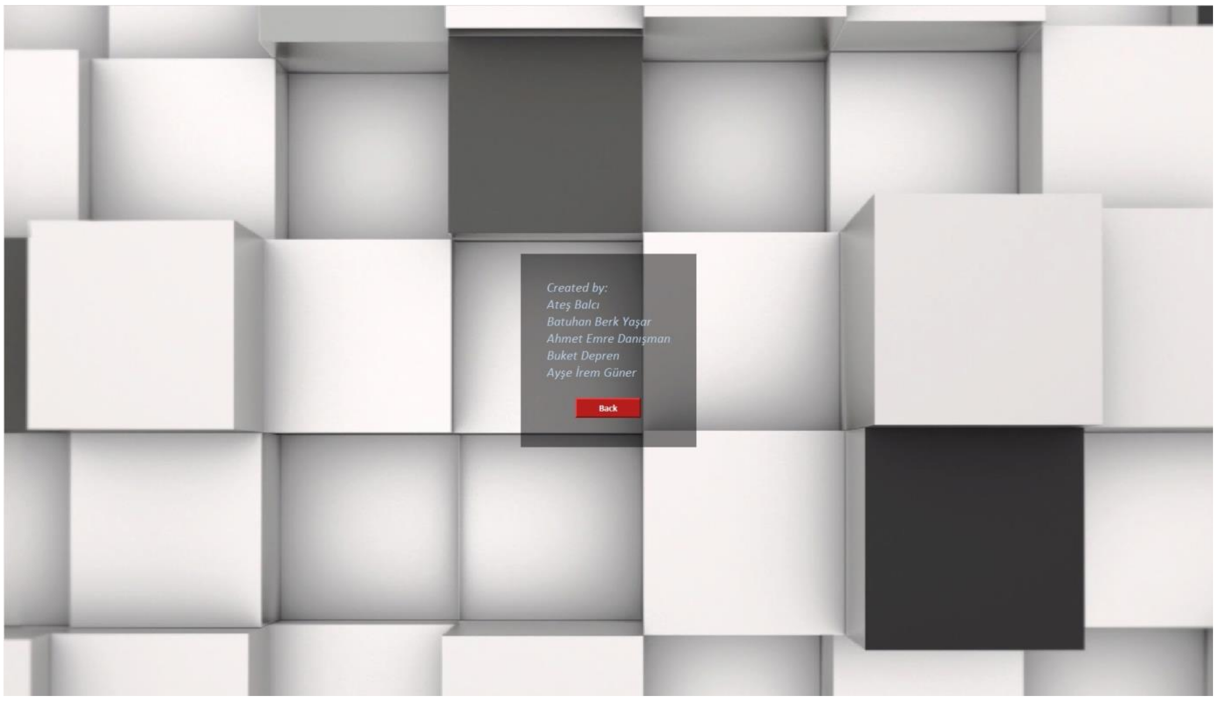
PlayerTrigger: Various sized, intractable objects that are interacted only by the main character when enters its area.

3.7.6 Option Screen



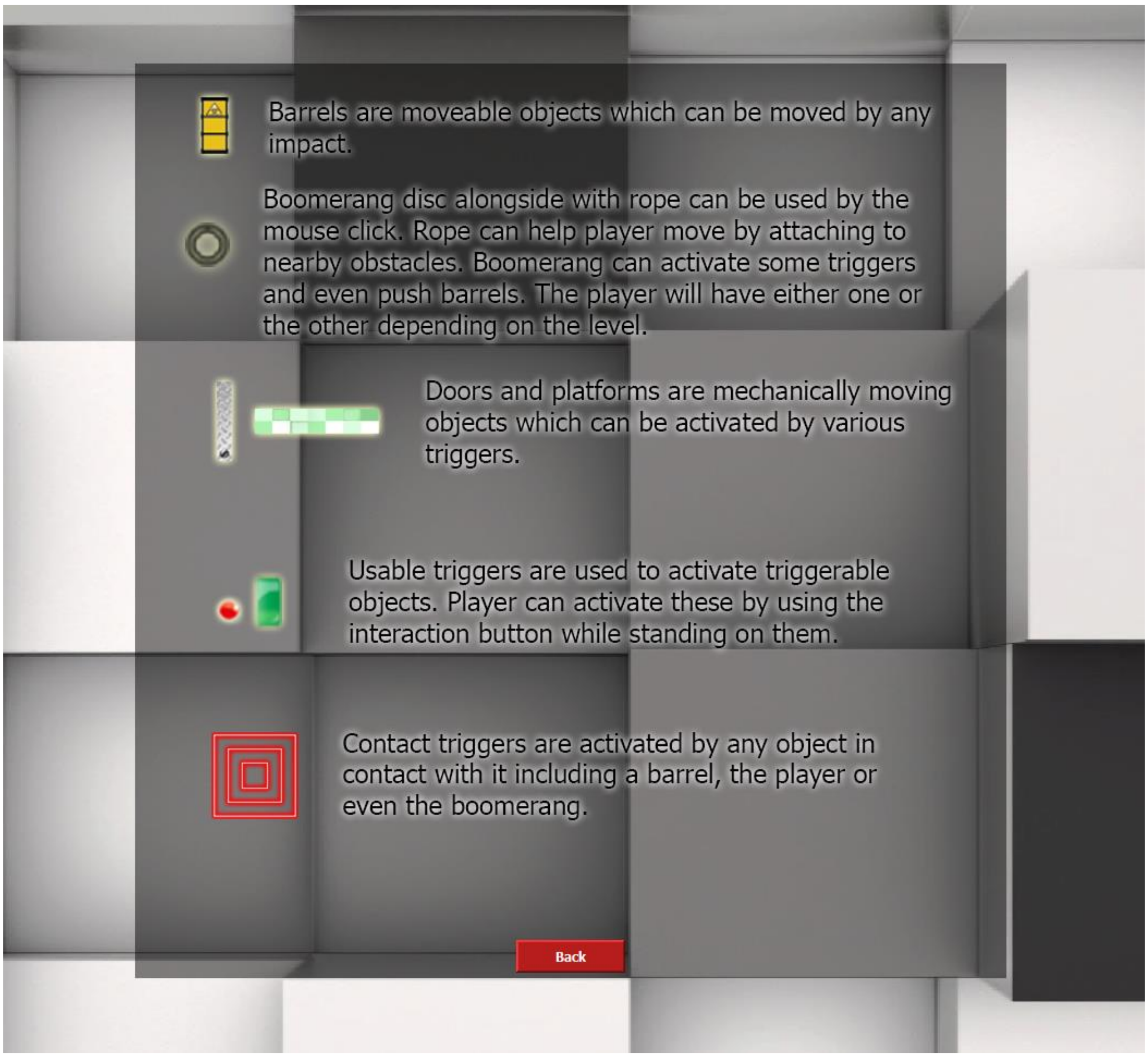
User can determine which buttons to use in the game to jump, go left, go right, interact with intractable objects, and restart the map. After changes are made, user can return to the main menu.

3.7.7 Credits Screen


Created by:
Ateş Balcı
Batuhan Berk Yaşar
Ahmet Emre Danışman
Buket Depren
Ayşe İrem Güner

Here, user can see developers and then return to the main menu.

3.7.8 Help Screen



Barrels are moveable objects which can be moved by any impact.

Boomerang disc alongside with rope can be used by the mouse click. Rope can help player move by attaching to nearby obstacles. Boomerang can activate some triggers and even push barrels. The player will have either one or the other depending on the level.

Doors and platforms are mechanically moving objects which can be activated by various triggers.

Usable triggers are used to activate triggerable objects. Player can activate these by using the interaction button while standing on them.

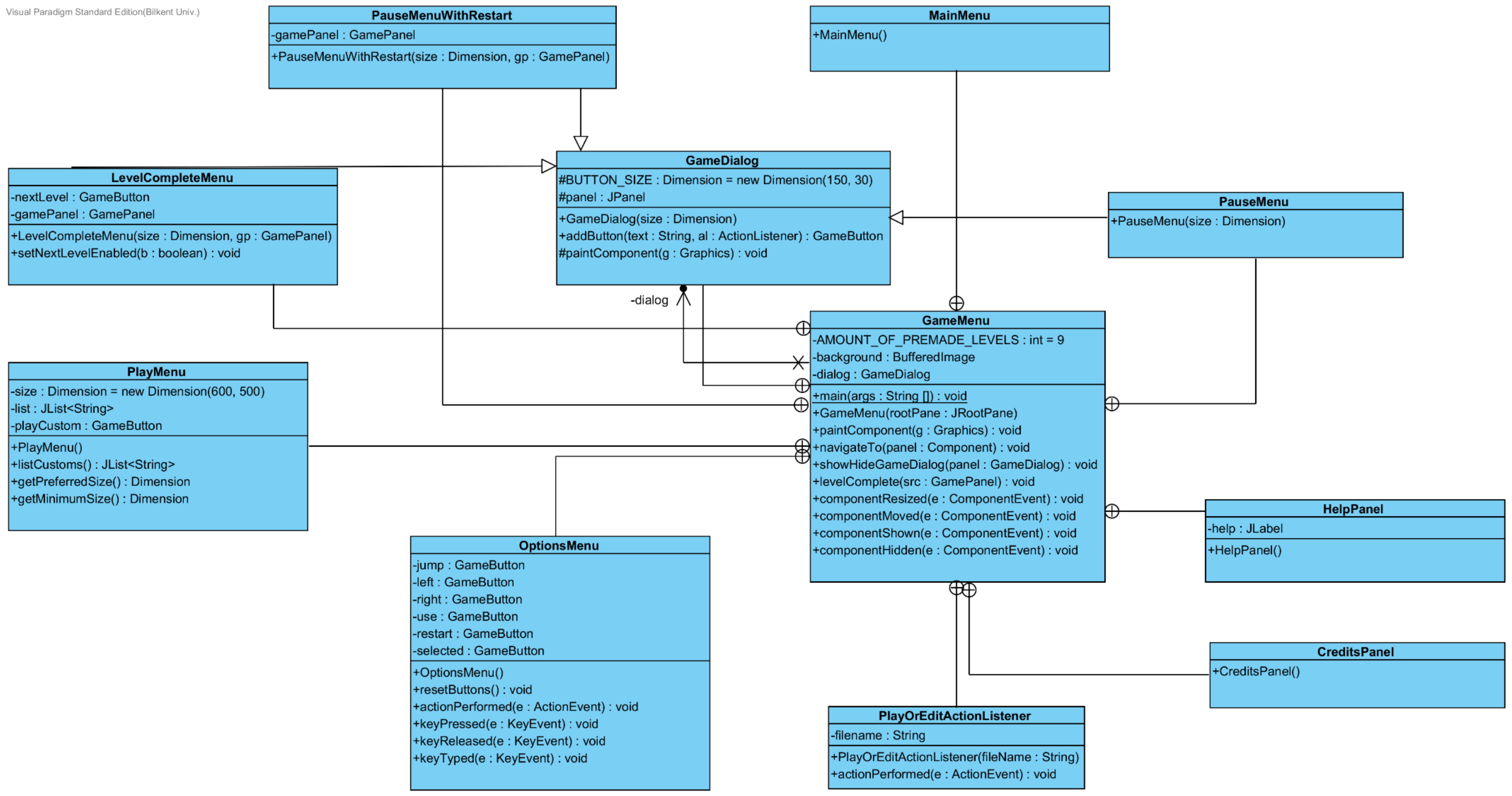
Contact triggers are activated by any object in contact with it including a barrel, the player or even the boomerang.

In this screen, there tips and hints that can help user during the gameplay.

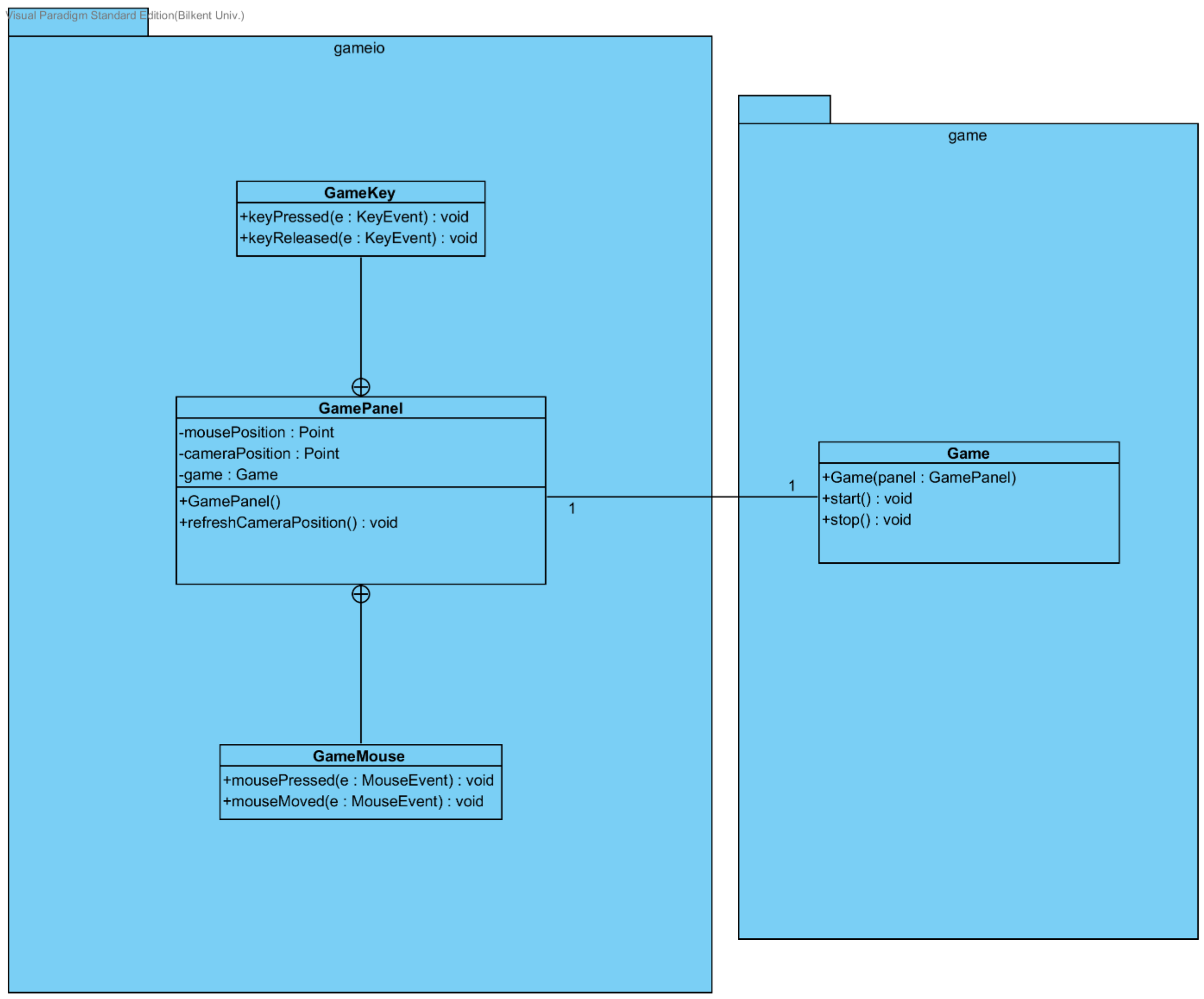
4 Analysis Models

4.1 Object Model

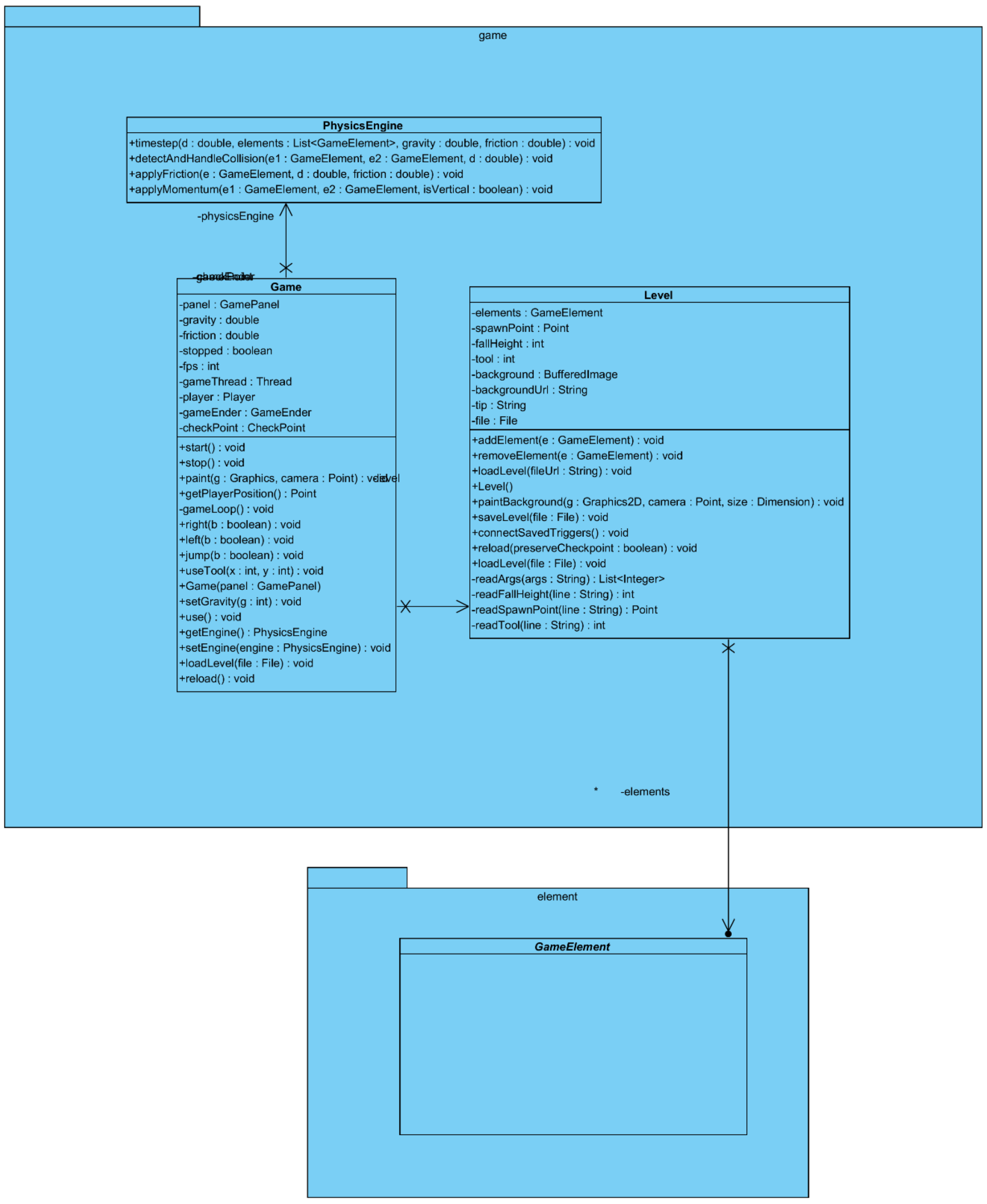
4.1.1 User Interface Management Subsystem


Visual Paradigm Standard Edition(Bilkent Univ.)
PauseMenuWithRestart
-gamePanel : GamePanel
+PauseMenuWithRestart(size : Dimension, gp : GamePanel)
MainMenu
+MainMenu()
GameDialog
#BUTTON_SIZE : Dimension = new Dimension(150, 30)
#panel : JPanel
+GameDialog(size : Dimension)
+addButton(text : String, al : ActionListener) : GameButton
#paintComponent(g : Graphics) : void
LevelCompleteMenu
-nextLevel : GameButton
-gamePanel : GamePanel
+LevelCompleteMenu(size : Dimension, gp : GamePanel)
+setNextLevelEnabled(b : boolean) : void
PauseMenu
+PauseMenu(size : Dimension)
-dialog
GameMenu
-AMOUNT_OF_PREMADE_LEVELS : int = 9
-background : BufferedImage
-dialog : GameDialog
+main(args : String []) : void
+GameMenu(rootPane : JRootPane)
+paintComponent(g : Graphics) : void
+navigateTo(panel : Component) : void
+showHideGameDialog(panel : GameDialog) : void
+levelComplete(src : GamePanel) : void
+componentResized(e : ComponentEvent) : void
+componentMoved(e : ComponentEvent) : void
+componentShown(e : ComponentEvent) : void
+componentHidden(e : ComponentEvent) : void
PlayMenu
-size : Dimension = new Dimension(600, 500)
-list : JList<String>
-playCustom : GameButton
+PlayMenu()
+listCustoms() : JList<String>
+getPreferredSize() : Dimension
+getMinimumSize() : Dimension
OptionsMenu
-jump : GameButton
-left : GameButton
-right : GameButton
-use : GameButton
-restart : GameButton
-selected : GameButton
+OptionsMenu()
+resetButtons() : void
+actionPerformed(e : ActionEvent) : void
+keyPressed(e : KeyEvent) : void
+keyReleased(e : KeyEvent) : void
+keyTyped(e : KeyEvent) : void
HelpPanel
-help : JLabel
+HelpPanel()
CreditsPanel
+CreditsPanel()
PlayOrEditActionListener
-filename : String
+PlayOrEditActionListener(fileName : String)
+actionPerformed(e : ActionEvent) : void

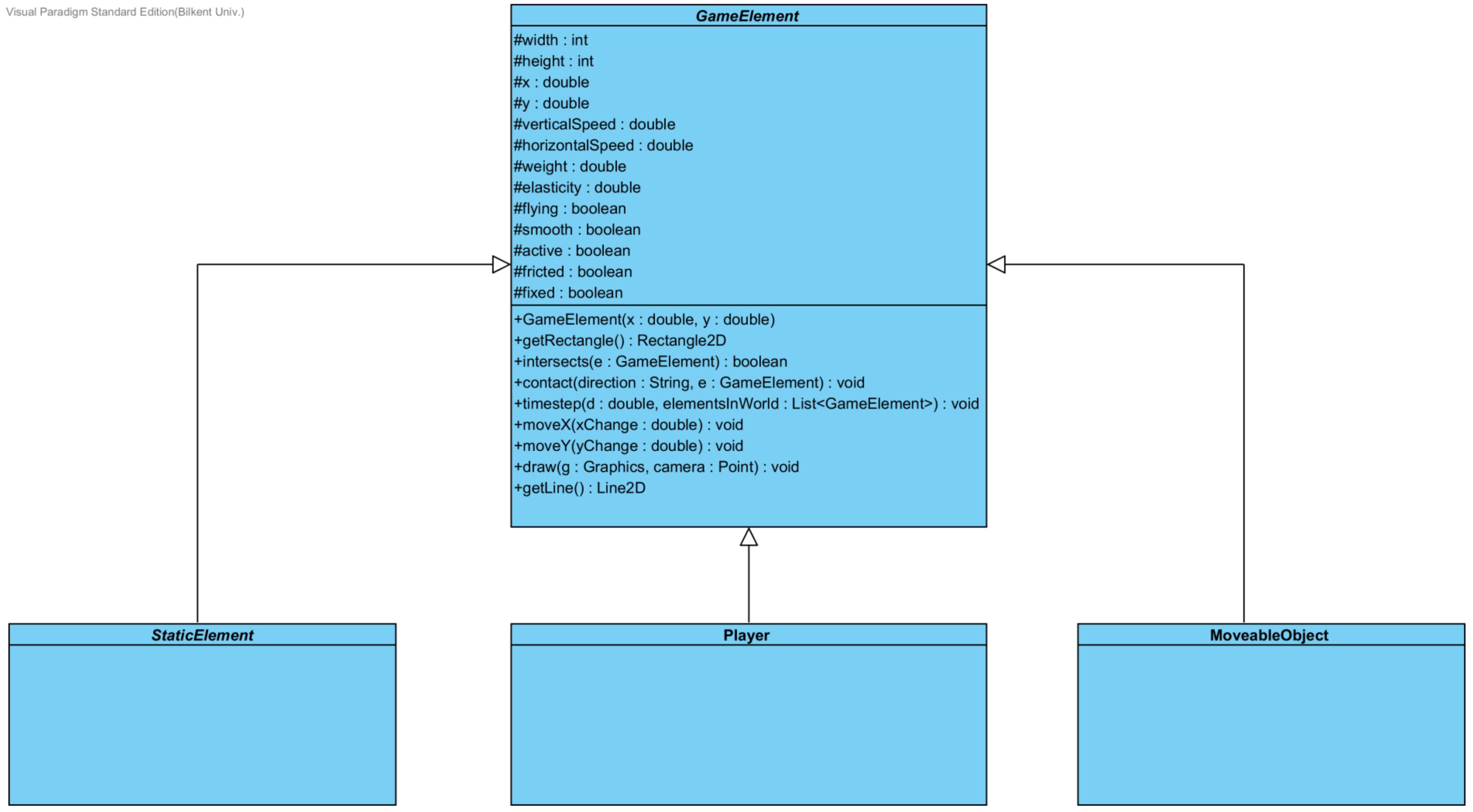
4.1.2 Game I/O Subsystem


Visual Paradigm Standard Edition(Bilkent Univ.)
gameio
GameKey
+keyPressed(e : KeyEvent) : void
+keyReleased(e : KeyEvent) : void
GamePanel
-mousePosition : Point
-cameraPosition : Point
-game : Game
+GamePanel()
+refreshCameraPosition() : void
GameMouse
+mousePressed(e : MouseEvent) : void
+mouseMoved(e : MouseEvent) : void
1
1
game
Game
+Game(panel : GamePanel)
+start() : void
+stop() : void

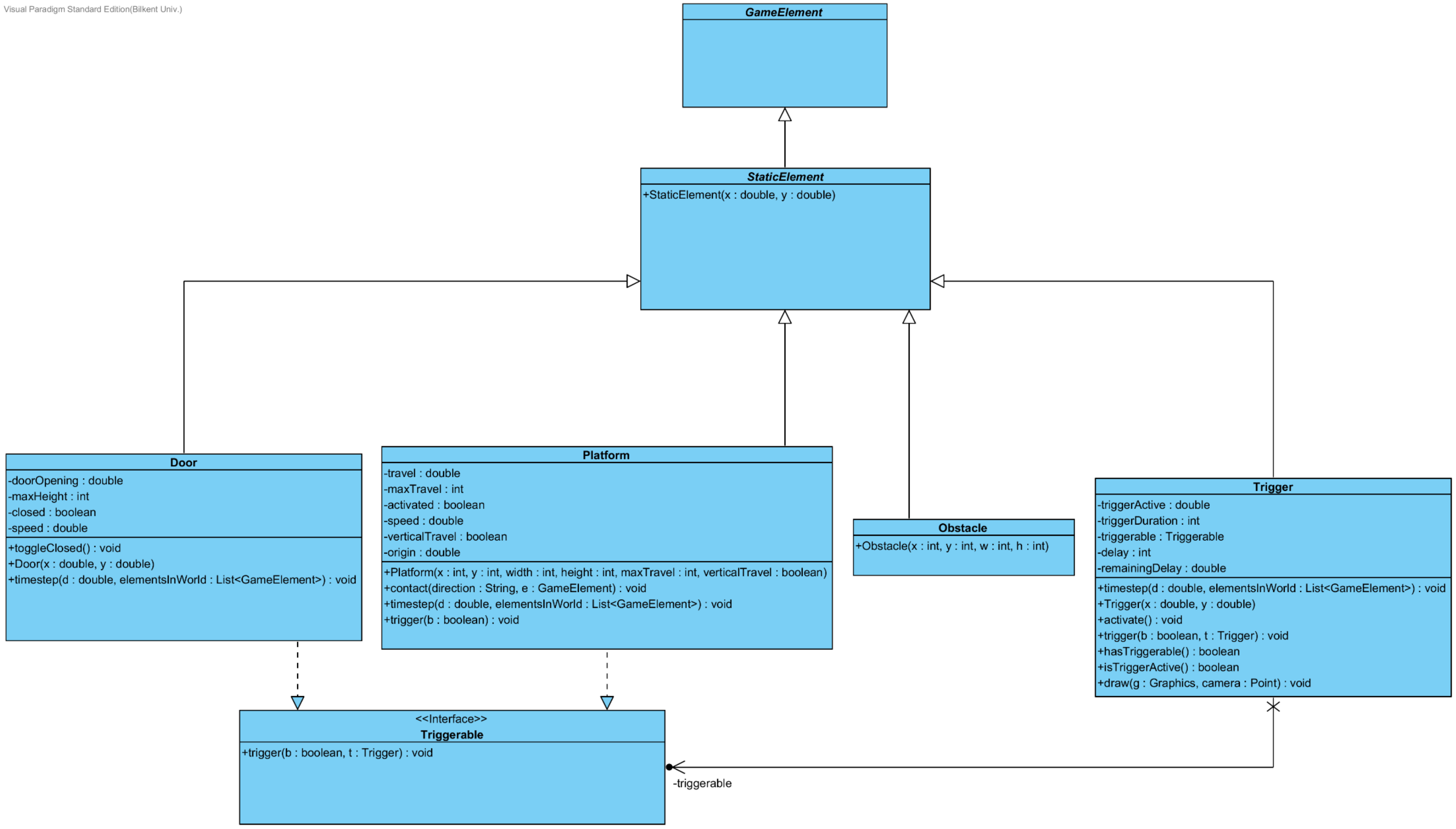
4.1.3 Game Logic Subsystem


game
PhysicsEngine
+timestep(d : double, elements : List<GameElement>, gravity : double, friction : double) : void
+detectAndHandleCollision(e1 : GameElement, e2 : GameElement, d : double) : void
+applyFriction(e : GameElement, d : double, friction : double) : void
+applyMomentum(e1 : GameElement, e2 : GameElement, isVertical : boolean) : void
-physicsEngine
Game
-panel : GamePanel
-gravity : double
-friction : double
-stopped : boolean
-fps : int
-gameThread : Thread
-player : Player
-gameEnder : GameEnder
-checkPoint : CheckPoint
+start() : void
+stop() : void
+paint(g : Graphics, camera : Point) : void
+getPlayerPosition() : Point
-gameLoop() : void
+right(b : boolean) : void
+left(b : boolean) : void
+jump(b : boolean) : void
+useTool(x : int, y : int) : void
+Game(panel : GamePanel)
+setGravity(g : int) : void
+use() : void
+getEngine() : PhysicsEngine
+setEngine(engine : PhysicsEngine) : void
+loadLevel(file : File) : void
+reload() : void
-level
Level
-elements : GameElement
-spawnPoint : Point
-fallHeight : int
-tool : int
-background : BufferedImage
-backgroundUrl : String
-tip : String
-file : File
+addElement(e : GameElement) : void
+removeElement(e : GameElement) : void
+loadLevel(fileUrl : String) : void
+Level()
+paintBackground(g : Graphics2D, camera : Point, size : Dimension) : void
+saveLevel(file : File) : void
+connectSavedTriggers() : void
+reload(preserveCheckpoint : boolean) : void
+loadLevel(file : File) : void
-readArgs(args : String) : List<Integer>
-readFallHeight(line : String) : int
-readSpawnPoint(line : String) : Point
-readTool(line : String) : int
*
-elements
element
GameElement

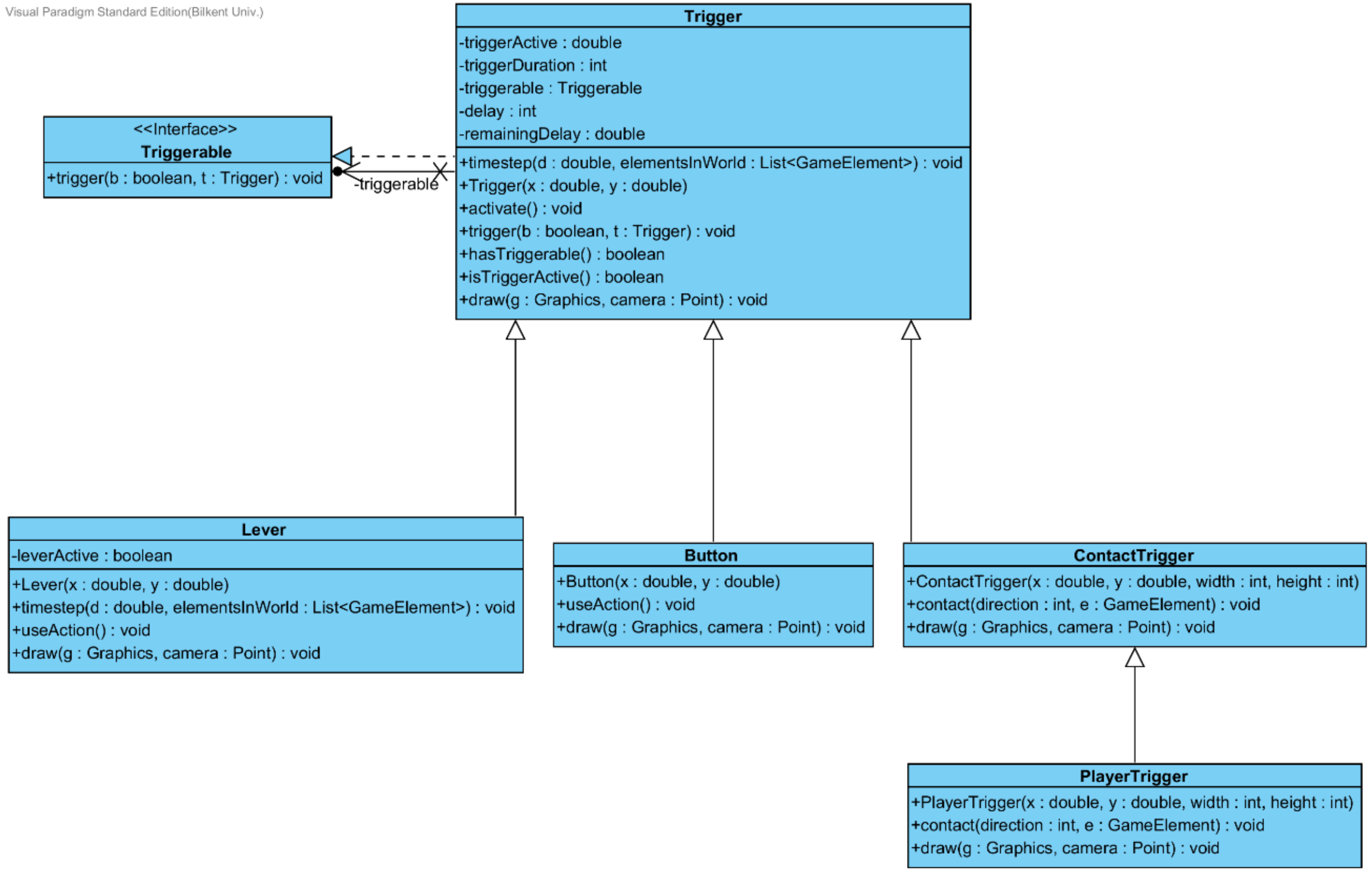
4.1.4 Game Elements Subsystem


Visual Paradigm Standard Edition(Bilkent Univ.)
GameElement
#width : int
#height : int
#x : double
#y : double
#verticalSpeed : double
#horizontalSpeed : double
#weight : double
#elasticity : double
#flying : boolean
#smooth : boolean
#active : boolean
#fricted : boolean
#fixed : boolean
+GameElement(x : double, y : double)
+getRectangle() : Rectangle2D
+intersects(e : GameElement) : boolean
+contact(direction : String, e : GameElement) : void
+timestep(d : double, elementsInWorld : List<GameElement>) : void
+moveX(xChange : double) : void
+moveY(yChange : double) : void
+draw(g : Graphics, camera : Point) : void
+getLine() : Line2D
StaticElement
Player
MoveableObject

4.1.4.1 StaticElement and Triggers


GameElement
StaticElement
+StaticElement(x : double, y : double)
Door
-doorOpening : double
-maxHeight : int
-closed : boolean
-speed : double
+toggleClosed() : void
+Door(x : double, y : double)
+timestep(d : double, elementsInWorld : List<GameElement>) : void
Platform
-travel : double
-maxTravel : int
-activated : boolean
-speed : double
-verticalTravel : boolean
-origin : double
+Platform(x : int, y : int, width : int, height : int, maxTravel : int, verticalTravel : boolean)
+contact(direction : String, e : GameElement) : void
+timestep(d : double, elementsInWorld : List<GameElement>) : void
+trigger(b : boolean) : void
Obstacle
+Obstacle(x : int, y : int, w : int, h : int)
Trigger
-triggerActive : double
-triggerDuration : int
-triggerable : Triggerable
-delay : int
-remainingDelay : double
+timestep(d : double, elementsInWorld : List<GameElement>) : void
+Trigger(x : double, y : double)
+activate() : void
+trigger(b : boolean, t : Trigger) : void
+hasTriggerable() : boolean
+isTriggerActive() : boolean
+draw(g : Graphics, camera : Point) : void
<<Interface>>
Triggerable
+trigger(b : boolean, t : Trigger) : void
-triggerable

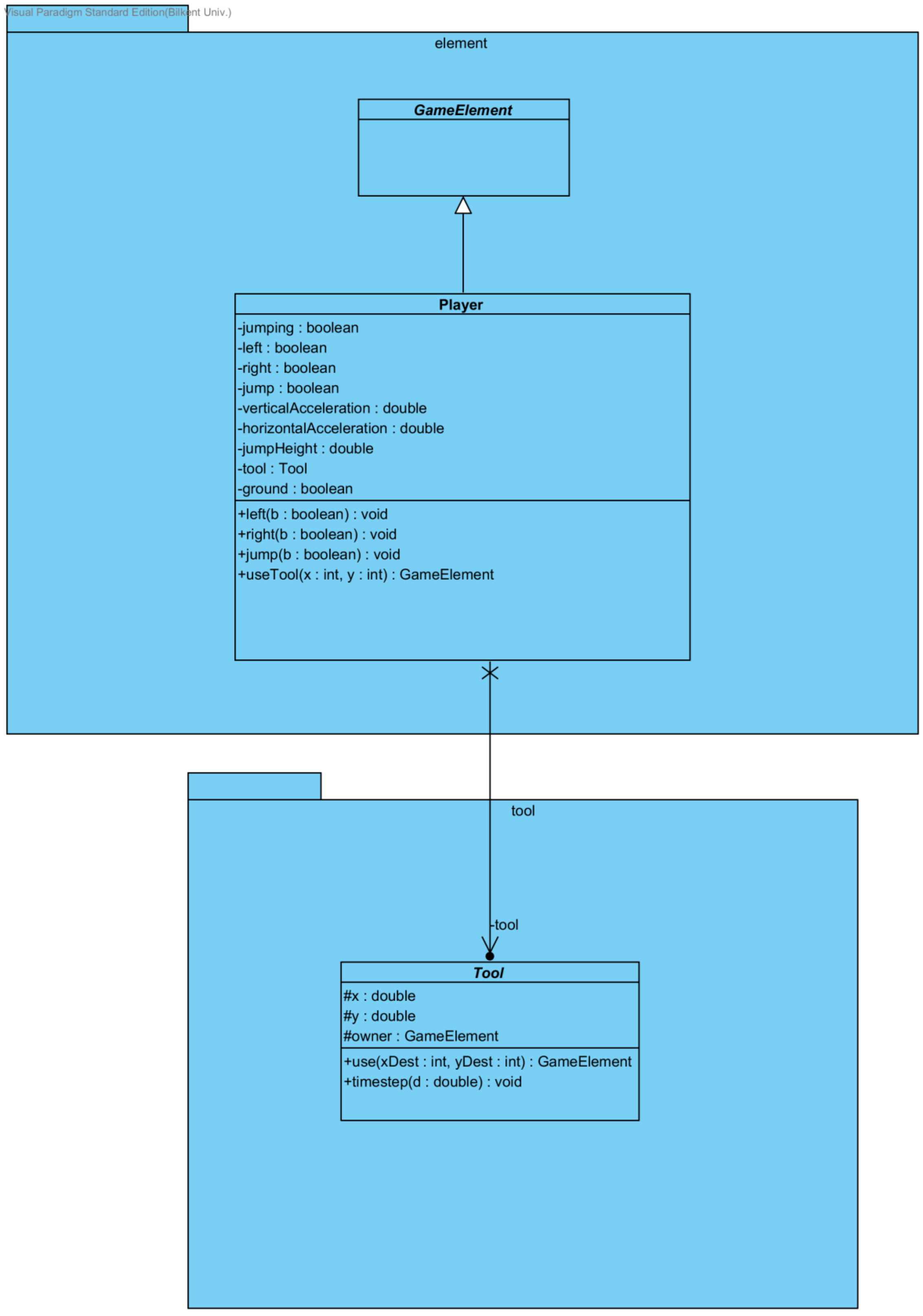
4.1.4.2 Triggers



Visual Paradigm Standard Edition(Bilkent Univ.)
<<Interface>>
Triggerable
+trigger(b : boolean, t : Trigger) : void
-triggerable
Trigger
-triggerActive : double
-triggerDuration : int
-triggerable : Triggerable
-delay : int
-remainingDelay : double
+timestep(d : double, elementsInWorld : List<GameElement>) : void
+Trigger(x : double, y : double)
+activate() : void
+trigger(b : boolean, t : Trigger) : void
+hasTriggerable() : boolean
+isTriggerActive() : boolean
+draw(g : Graphics, camera : Point) : void
Lever
-leverActive : boolean
+Lever(x : double, y : double)
+timestep(d : double, elementsInWorld : List<GameElement>) : void
+useAction() : void
+draw(g : Graphics, camera : Point) : void
Button
+Button(x : double, y : double)
+useAction() : void
+draw(g : Graphics, camera : Point) : void
ContactTrigger
+ContactTrigger(x : double, y : double, width : int, height : int)
+contact(direction : int, e : GameElement) : void
+draw(g : Graphics, camera : Point) : void
PlayerTrigger
+PlayerTrigger(x : double, y : double, width : int, height : int)
+contact(direction : int, e : GameElement) : void
+draw(g : Graphics, camera : Point) : void

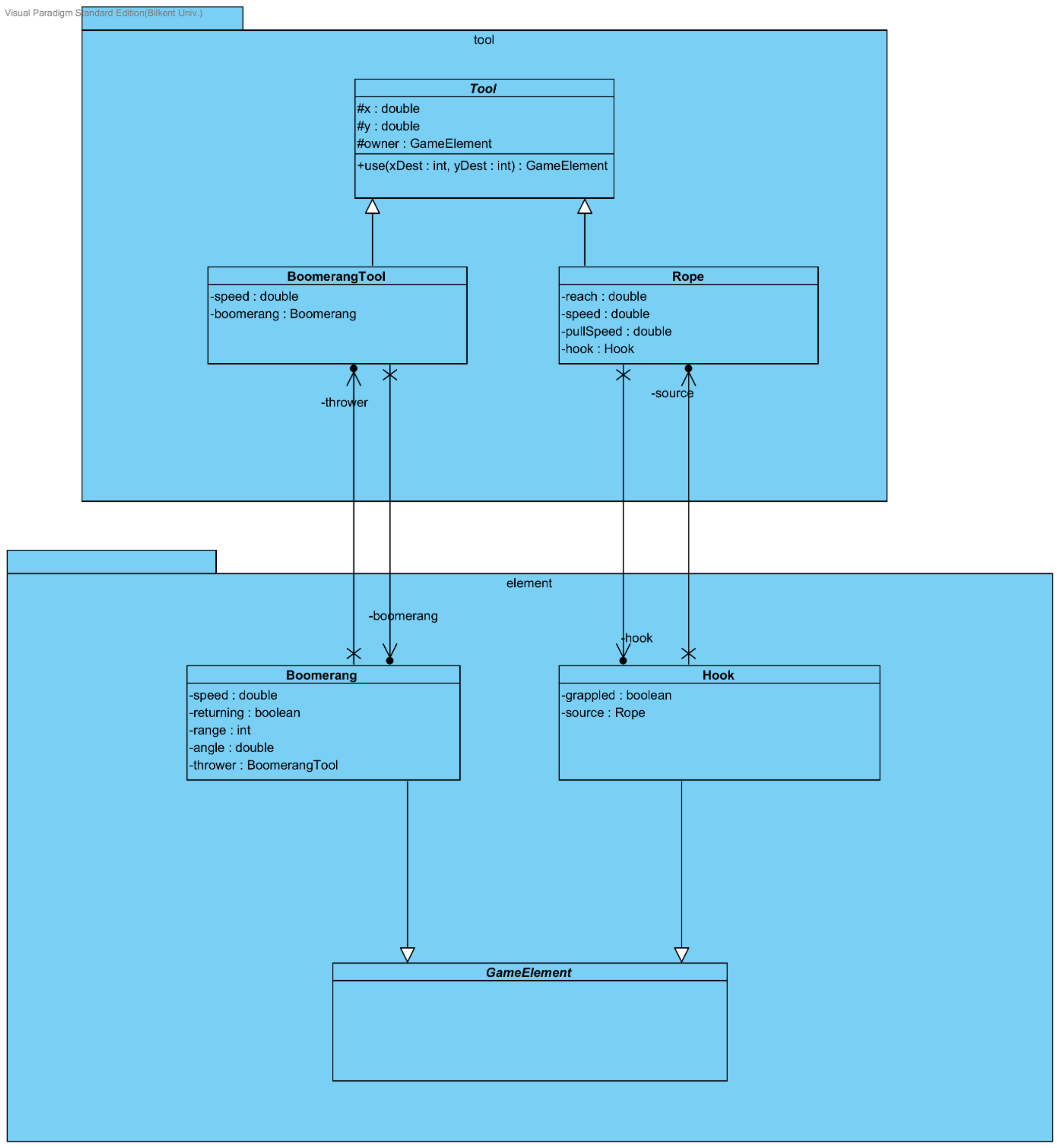
4.1.5 Player and Tools Subsystem

4.1.5.1 Player Class



Visual Paradigm Standard Edition(Bilkent Univ.)
element
GameElement
Player
-jumping : boolean
-left : boolean
-right : boolean
-jump : boolean
-verticalAcceleration : double
-horizontalAcceleration : double
-jumpHeight : double
-tool : Tool
-ground : boolean
+left(b : boolean) : void
+right(b : boolean) : void
+jump(b : boolean) : void
+useTool(x : int, y : int) : GameElement
tool
-tool
Tool
#x : double
#y : double
#owner : GameElement
+use(xDest : int, yDest : int) : GameElement
+timestep(d : double) : void

4.1.5.2 Tool Class



4.2 Dynamic Models

4.2.1 State Chart

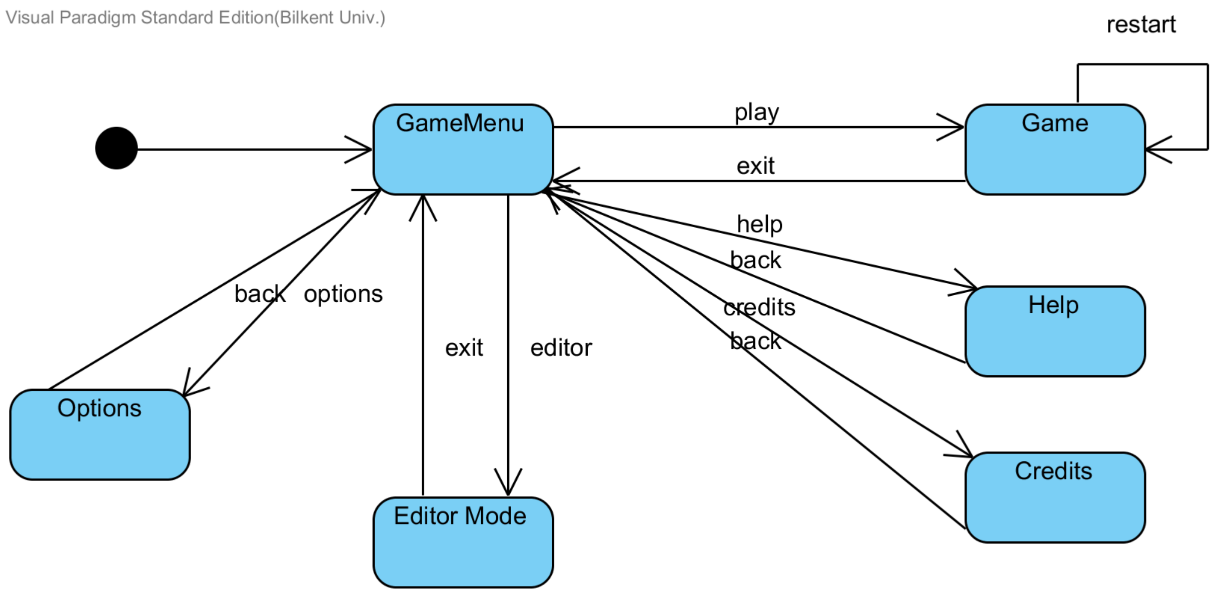
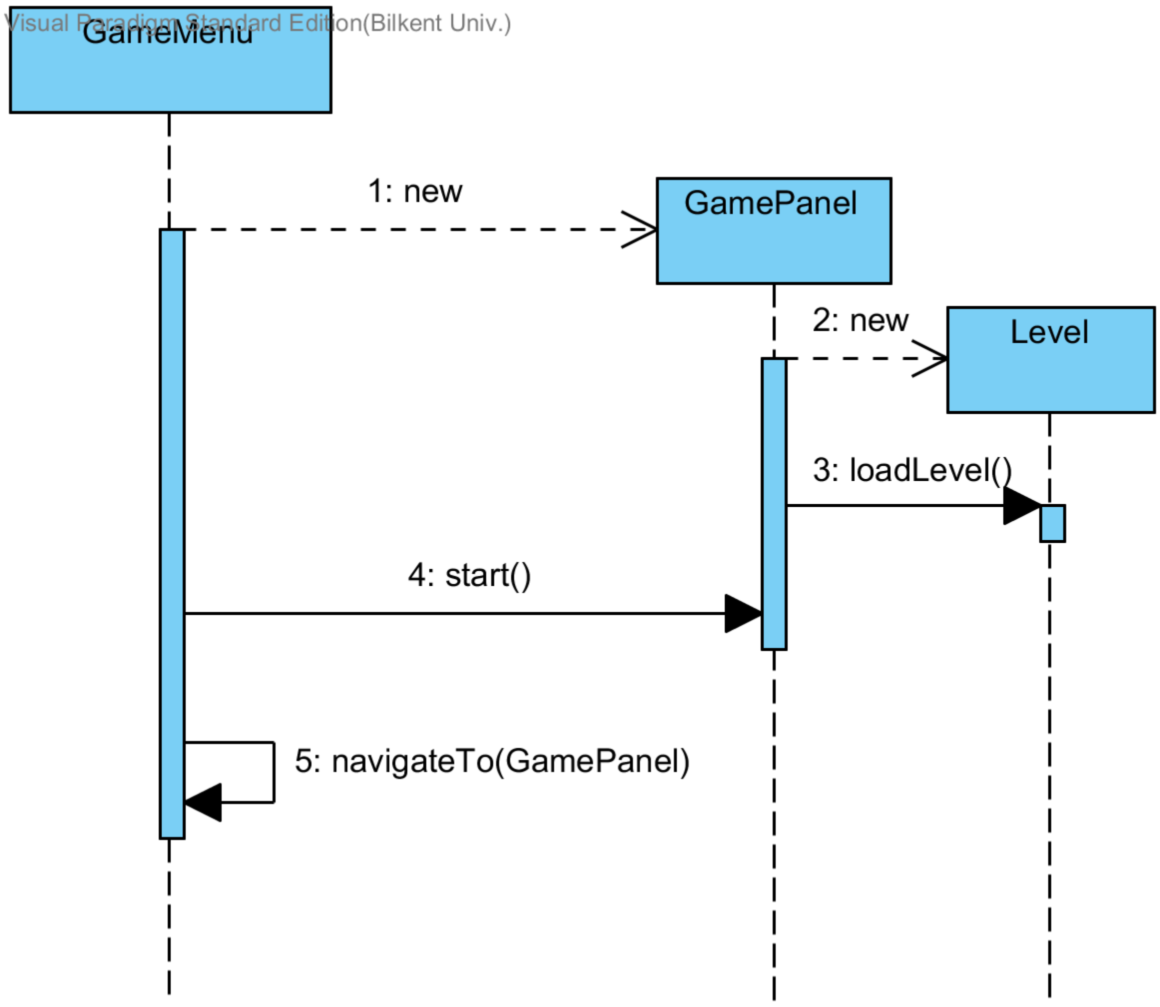


Figure shows the state diagram of the GameMenu navigation class.

4.2.2 Sequence Diagrams

4.2.2.1 Starting to Play a Level



This diagram shows how the GamePanel class is instantiated and the saved level file on the hard drive is loaded. The work is actually done by Game class but it is handled by its ui class which is GamePanel in this case.

4.2.2.2 Running of the Game

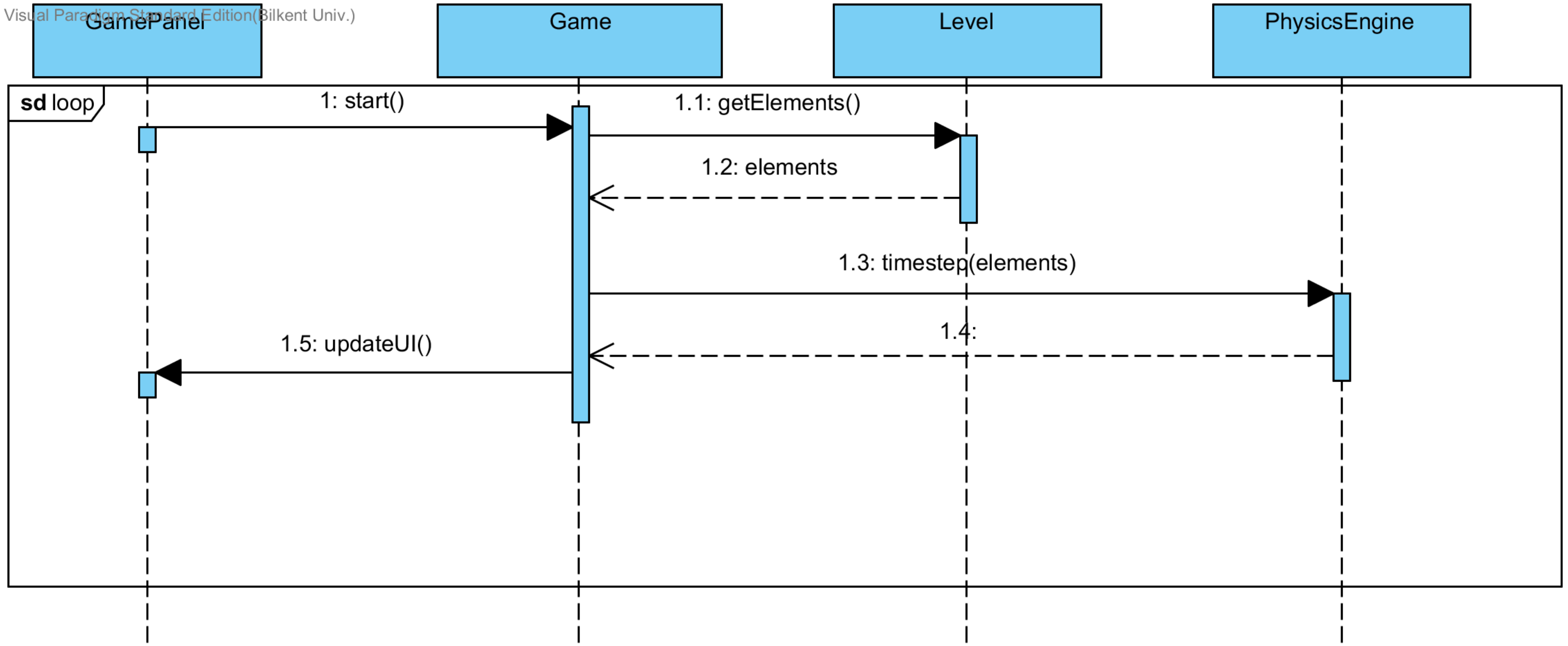


Figure shows the GamePanel giving the Game class the starting signal. After that method call, Game class runs the timestep algorithm and queues repaint event so that the screen is refreshed.

5 System Design

5.1 Design Goals

Before the coding part had been started, design goals were decided in order to create a game with optimum quality and plan how to implement the game. There were some system features that were effective on this process. The most important ones can be seen below.

5.1.1 Reliability

Dungeon Escape is a bug-free and non-crashing program which does not lose any data run consistently thanks to the language it was implemented in, Java. The other factor that helps Dungeon Escape to be a reliable system is its number of input buttons. Since there are not too many of them, the system does not use too much space on stack memory and is unlikely to have out-of-memory related problems. So, the system will give the same result to the same move, anytime it is executed. Also, there will be checkpoints in game's some levels so that player will not lose any in-game progress. After these features are decided and implemented, a lot of testing on different systems had been done to test the game's reliability.

5.1.2 Extensibility

Dungeon Escape is designed for regular people to play, as well as hardcore gamers. In order to get regular people to extend, modify and play the game to its fullest capacity, an easy to use 'Editor Mode' is added to game's main menu. Any person can run this editor and design his/her own levels using all the tools available in the game and play the designed levels afterwards. These new levels will give the game a considerable depth, but will not have a major effect on gameplay. Editor mode is a mind blowing tool for regular people to expand the game, but hardcore gamers may not be so impressed by this tool. Since Dungeon Escape is an object-oriented programmed game, it is always possible to modify existing features (e.g. momentum, tools, gravity) or add new ones by simply creating new classes. By this method, the game can be changed vastly and become a whole new game. Another and easier to modify the game is changing its background, object images or sounds. Users can access those files and by editing them, the game can be modified even more. With all these features, all types of players' interest and excitement about the game can be sustained.

5.1.3 Performance

As anyone that has played game at least once knows, lagging and freezing are not fun and since the main aim of playing a game is to have fun, high performance is a must in any game. These easy-to-recognize problems may cause the user to not play the game, despite of all other great features that the game has. In order to prevent these problems, the game's performance must be high enough to keep the player entertained. Dungeon Escape's response to the player will be immediate and will run on at least 60 FPS so that the game will be fluent all the time and will not bother the player with performance related problems. In order to achieve 60 FPS goal, Dungeon Escape is using the computer's graphic card to generate visuals instead of processor built-in graphic card.

5.1.4 Usability

If developers want a game to be played, they should design an easy to play and fully understandable game and to do so, it should be user friendly so that players will not be have to focus on learning the game, instead they can focus on mastering it. To accomplish usability criteria, players should not have trouble with interacting items, using tools, recognizing map etc. First of all, the user will be familiar with the game's elements via the 'Help' screen which is accessible from the main menu. This help menu will help user to get familiar with the game elements which they will come across. Also, every single object in the game is identical which helps user to distinguish them easily. On top of that, triggerable objects look different to be distinguished quickly while solving puzzles. If the user is not comfortable with default key assignments, can change the keyboard control buttons in the options menu. Even there are all these helps, if user still fails to finish a level, levels will start from the last checkpoint to make levels more comfortable for the user. The main menu and pause menu are plain-themed and controlled by mouse, which helps user to navigate in it without a problem.

5.1.5 Portability

Considering performance and running speed of a program, Java may not be the best programming language choice to code a game, but we have wanted Dungeon Escape to be a cross-platform game that anyone can play without dealing with any system requirement problems. It is an important feature that a game should have if developers want it to reach a wide range of users. In this aspect, Java is superior compared to other programming languages that pledge better performance, like C++. As developers of Dungeon Escape, we believe that portability versus performance is a reasonable tradeoff because our game is not too complex so that it does not need a heavy performance structure. Yet, it works with 60 FPS always, which indicates that we have made a good tradeoff.

5.2 Sub-System Decomposition

In Dungeon Escape, there are 2 main subsystems: GameElement and Tool. There 3 also have their own subsystems.

5.2.1 GameElement Subsystem

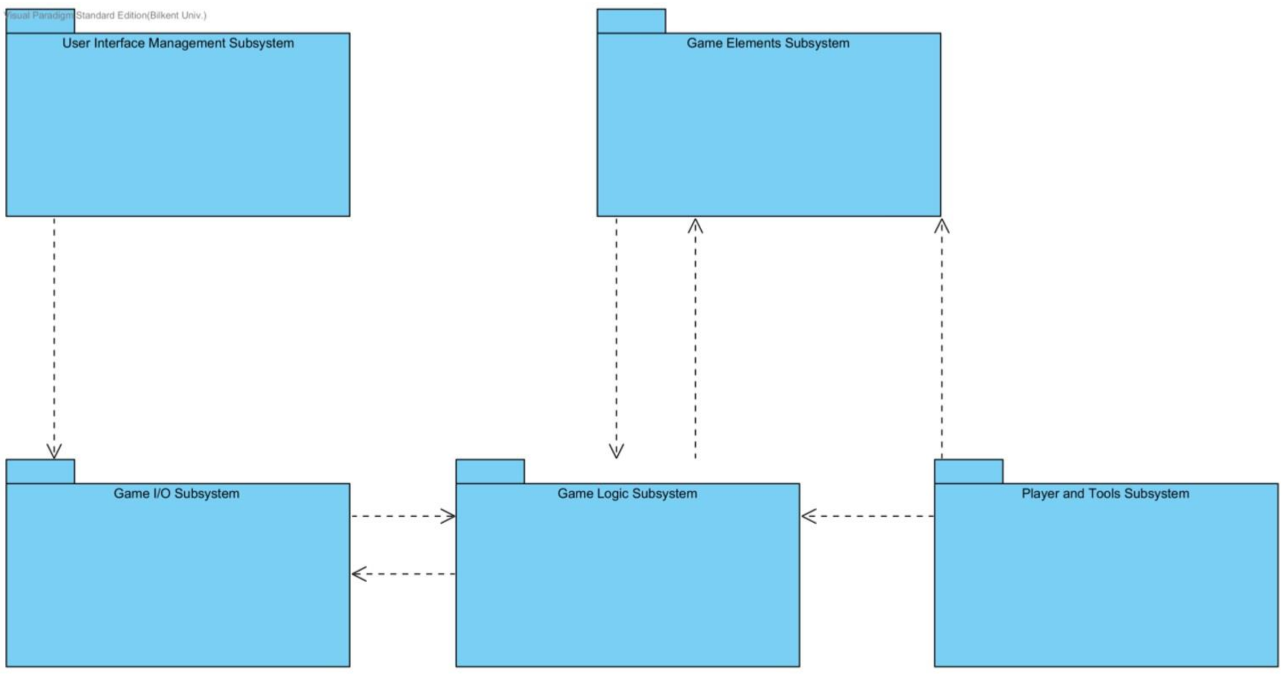
This class is an abstract class and can’t be instantiated by itself. All elements with physical presence on a level extend this class to be implemented. GameElement subsystem consists of very object in the game because they all have to extend it.

5.2.2 Tool Subsystem

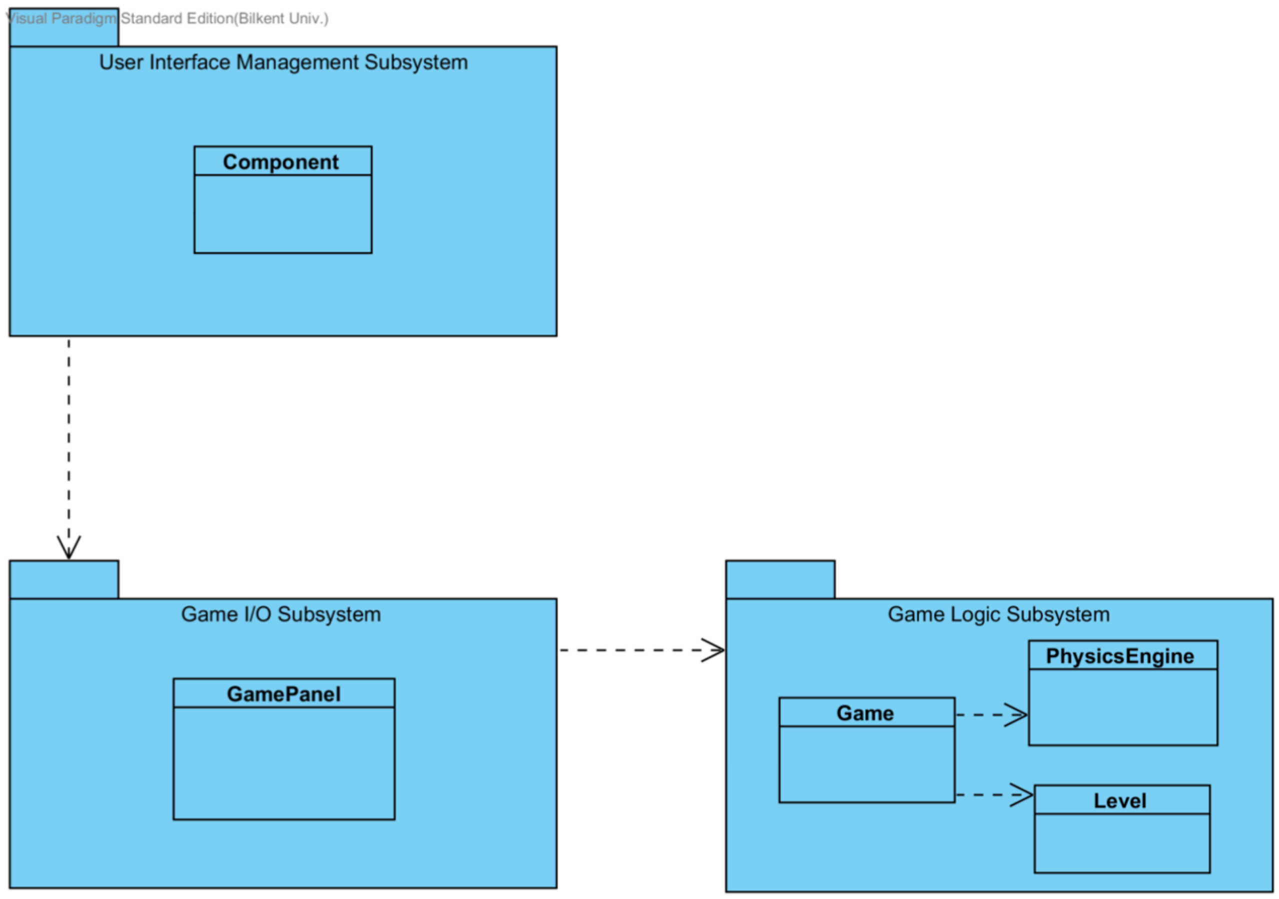
This abstract class includes the tools that main character uses in the game and enables the user to utilize mechanics other than the regular game mechanics such as running and jumping. This class can be extended in order to create new tools for main character, give new abilities to the user and mechanics to the game.

5.3 Architectural Patterns

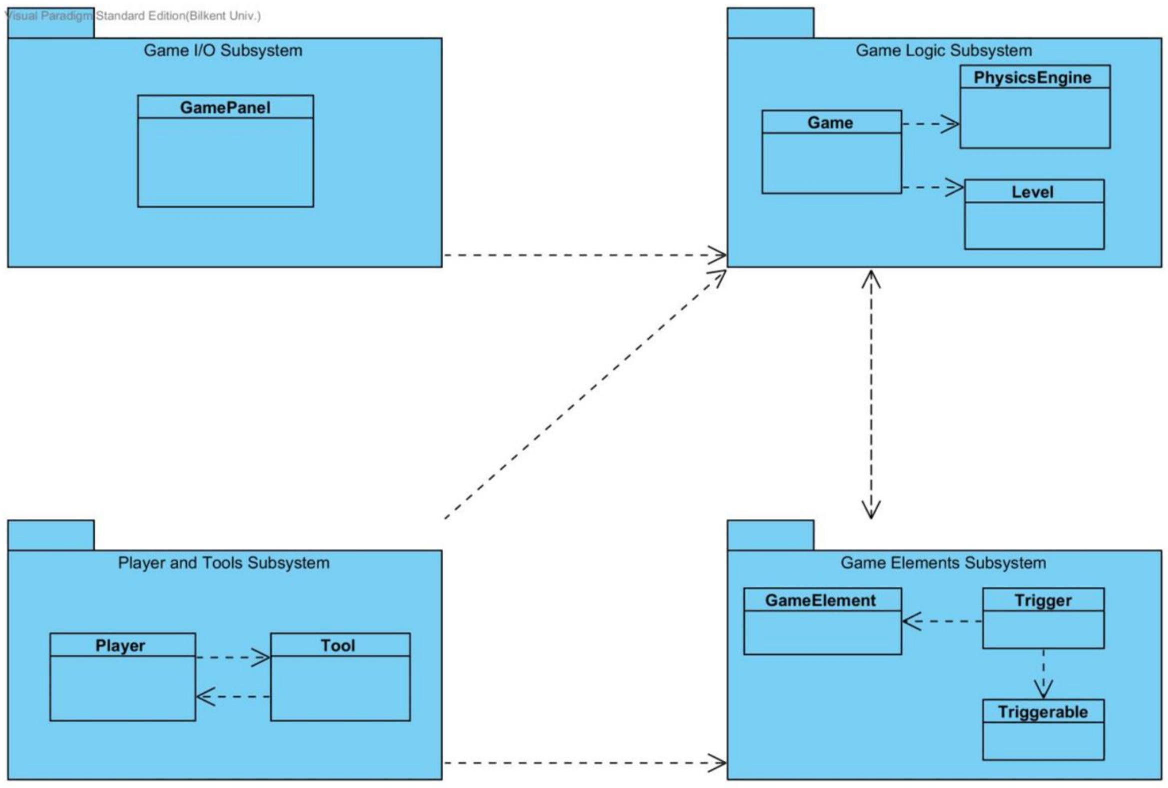
Among all architectural styles, we find out that the most suitable style is three-tier architecture. Since we have neither server nor database in our system, it only has Client and Controller components. Because of that three-tier is the best architectural style for our system. You can find the diagram describing three-tier principle of our design is depicted in the following figure.


Visual Paradigm Standard Edition(Bilkent Univ.)
User Interface Management Subsystem
Game Elements Subsystem
Game I/O Subsystem
Game Logic Subsystem
Player and Tools Subsystem

In presentation layer, we have classes that provide an interface to user for accessing the system. The first interaction of the user with system starts at the “Component” class, which manages the choice of Menu type. “Component” will choose whether the submenu or the menu is desired and it will continue to do its operations. Afterwards, it delegates user’s choices to Game I/O and Game Logic Subsystems.


Visual Paradigm Standard Edition(Bilkent Univ.)
User Interface Management Subsystem
Component
Game I/O Subsystem
GamePanel
Game Logic Subsystem
PhysicsEngine
Game
Level

In the logic tier, the input of the user is evaluated in the “Game I/O Subsystem”. The evaluation of the GIOS directs to the “Game” class in the “Game Logic Subsystem.” GLS constructs the game layout by using the “PhysicsEngine” and “Level” classes. During the construction process of the GLS, the “Game” which is Façade class of the GLS interacts with GIOS to check the user inputs. After the logic is constructed, the system will direct to the Player and Tools Subsystem & Game Elements Subsystem to update its features.



In the operation layer, Player and Tool Subsystem determine what the user can or cannot do during the game play. “Player” class which is inside the PTS, sets the abilities of the game character such as, jump height, running speed, etc. “Tool” class contains the items that the game character has and the operations that can be done by these items. Player class in PTS extends the “GameElement” class in Game Elements Subsystem. Inside the GES there are two classes which define the structures and styles of the objects in the game. “Trigger” class and “Triggrable” interface bring dynamism to the game. In other words, with the help of these instances, the interactivity of the objects will be extended.

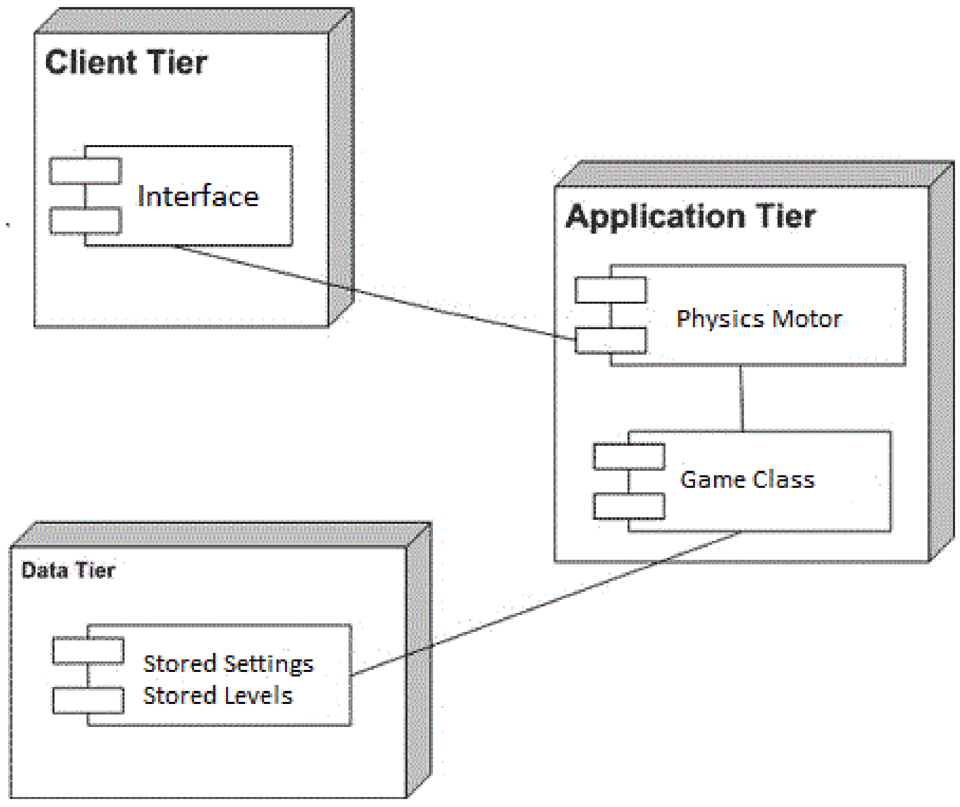
Another architectural pattern that can be mentioned is server client pattern. Dungeon Escape’s implementation was not decided in order to make it compatible with server client pattern, but the way Editor Mode is coded is coherent with it. Editor mode allows users to create their own levels, save them to game’s directory and play them later.

While the computer itself is powerful, presentation layers of editor moves to the client. Client performs some service in the editing screen, and editor responds to them.

5.4 Hardware Software Mapping

Dungeon Escape is a standalone application, which will not require any kind of web or network system to operate. In order to run the game, any standard desktop pc’s configuration will be enough because it does not use any additional libraries such as DirectX libraries. Being standalone application, once Dungeon Escape is installed to the hard drive, its subsystems will be mapped to hard drive, memory and graphics card. During runtime, objects will be created and used in memory while data files (.level, mp3, .png, .jpg files) resides in hard drive. Other than that, graphical context (Textures, animations) will be executed in GPU. Java’s 2D graphics library (Graphics2D Class) uses GPU. In software mapping part, Dungeon Escape is implemented in Java programming language alongside with graphics 2D library we used wt and swing libraries of Java.

In Dungeon Escape, mouse and keyboard will be used as hardware configuration. Mouse will be used in the menus and during gameplay both to aim and change camera angle. Keyboard will be used to move, interact with objects, restart the game and open the menu during gameplay. Also, both mouse and keyboard will be used in Editor Mode in order to create a new level easily. Also, to hear sound effects of the game, an earphone or speaker is essential for Dungeon Escape.



5.5 Adressing Key Concerns

5.5.1 Persistent Data Management

There will be no setup process in order to paly Dungeon Escape and all the game data will be stored on the user's hard disk drive. Data is kept in files will be saved locally(e.g. custom levels, images, sounds) in files instead of database. In this project, database system is not required because it creates too much complexity and there is not extreme data flow because we deal with the data flow issue by keeping the data in user's hard drive. Therefore, game will access to stored data faster and will avoid unnecessary complexity in the memory. Any data that is used in Dungeon Escape will be available in the local files and users will be able, also encouraged, to modify visuals (e.g. background or object's images) and sound files.

5.5.2 Access Control and Security

Dungeon Escape is an offline single player game, which means only one user will interact with it on every PC. Also, there is not any data that belongs to a specific user and should be protected. So, it is not needed of designing a registration, logging in stage for the game. Also, image and sound files that game uses are accessible because making Dungeon Escape a easily modifiable game is one of our main priorities. In order to make this modifying process easier and faster for regular people that play our game, files are available and there is no protection.

5.5.3 Global Software Control

Global control flow is implemented in event-driven way. Program consistently waits for an event, a key stroke on the keyboard, mouse click or any mouse movement; and updates the views. 'Game' class determines any input and gives the appropriate answer to the user via changing game visuals. The control is more decentralized because different objects decide on the actions by evaluating different events. There is no "spider" or main controller that knows everything and controls the flow of all the data. This choice of control is explained by, again, the need for performance. Because it is an interactive system, the need for high performance is supported by distributing the control and avoiding any possible bottlenecks.

5.5.4 Boundary Conditions

When initializing the game, main menu appears with game's logo on the top of the screen. User can select 'Play' to select a level and start playing, 'Editor' to enter editing mode and modify existing levels or create a new one, 'Options' to change default key bindings, 'Credits' to view credits and 'Exit' to leave the game.

If the game’s loading mechanic (level class) is unable to locate or load a level that is wanted to be played because of any reason (e.g. corrupted level file, missing documents), the game will run the default (empty) level until it is stopped by the user. Later, user can select another level and replace the corrupted one.

6 Object-Design

6.1 Design Patterns

6.1.1 Observer Pattern

Observer pattern is a behavioral pattern that is used when there is one to many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. In other words, the observer pattern is used to observe the changes in an object’s general state. Any other information is not required other than these changes.

In Dungeon Escape Project we used Observer Pattern in two different senses.

Firstly, we have the Trigerable class, which provides an interface two the Door and Platform objects. Both Platform and Door objects are triggerable.

A trigger is a non-smooth object by default. When a non-static element passes through this object, this object calls the trigger method of a set Triggerable. A Door object is a thin tall trigger object, which decreases height towards upwards when triggered. It may block the path of the player in game; player must find a way to trigger it to get through. A Platform object is also a trigger object, which is similar to Door, but it is more complex. It can be set to move any amount of pixels and its direction and axis can be set. Its timestep function is overridden to check for any non-fixed objects and drag them with it.

So, if there is any need to change these objects' state, Trigger objects can be modified and the Platform and Door objects will notify these changes accordingly. In other words, if there is any change in Trigger objects, the observer will notify these changes and update the Door and Platform objects' states.

Secondly, we used Observer Pattern for our Editor Mode. Our program provides a feature to the clients, which allows them to create their own game level. There are many different objects that can participate in level creation process. In order to realize the changes in the level design the Observer Pattern is applied to the Editor Mode. The properties of the objects such as the coordinates of the objects, size of the objects, background image of a level etc. can be set by the user in the Editor Mode. If there is a change in properties of these objects, Observer notifies that it should update the level design's state and save it.

6.1.2 Façade Pattern

The Façade Pattern is a structural design pattern that simplifies the interface to a complex system; because it is usually composed of all the classes, which make up the subsystems of the complex system. It provides methods to the system that does what is expected by system, using the methods in the libraries and packages.

Since the Façade Pattern provides a useful interface for the subsystems, we used this pattern in our subsystem decomposition.

In Game Logic Subsystem, which constructs most of the game design, Game Class is a Façade Class for the Physics Engine and Level classes. Since the Physics Engine and Level classes have complex properties inside of it, Game class contains common operations inside of it and provides an interface for these classes.

Figure: Game Class is a Façade Class of GLS which provides interface to the Physics Engine and Level Classes.

6.1.3 Abstract Factory Pattern

Abstract Factory Pattern provides a general understanding of the features rather than unnecessary details and future developments. By inheriting a common interface, similar user interfaces can be obtained with the Abstract Factory Pattern. The pattern provides a prototype to coder, which can be improved by adding specific features.

In Dungeon Escape, we have GameElement Class, which provides an interface to the Trigger and Triggerable classes. If we want to develop another type of object in our game like Trigger objects, we can use GameElement class to create it. By the help of the Abstract Factory Pattern in GameElement Class we can inherit the basic features of the new class from it and the new class will be completely created after adding the specific features.

Figure: GameElement Class is a Abstract Factory Class of Game Element Subsystem which provides interface to the Trigger class and possible future classes.

6.2 Class Interfaces

6.2.1 GamePanel Class

Display and input control is done by a subclass of JPanel called GamePanel. This class is responsible for displaying the elements within the game, calculate the offset of display (offset is dynamic depending on the position of the mouse cursor) and inform the game itself on any mouse click or keystroke.

refreshCameraPosition(): Calculates the position of the object drawing offset depending on the player’s position within the game and the position of the mouse cursor.

6.2.2 Game Class

This class is responsible for executing the physics engine on a set of GameElement objects on specific time intervals. It contains a Level object which contains a list of GameElements. GameElement objects are the building block of the levels within the game. Game class periodically calls the timeStep function of PhysicsEngine class which moves the GameElements one by one depending on their existing velocities and applies gravity and friction. PhysicsEngine also handles the collisions between objects and applies momentum on collisions.

start(): This method creates a new thread and calls the gameLoop method. Since this is a never ending loop until the game is paused or the level is completed, a new thread is necessary in order to keep the other functionalities of the program going.

gameLoop(): This method creates a loop which runs the timeStep method of PhysicsEngine class once every specific time period (60 times a second by default).

stop(): Stops the engine thread, can be started again by recalling start method.

6.2.3 PhysicsEngine Class

This class consists of a set of methods without any properties.

timestep(double): This method determines how the physics inside the game is going to work. It takes the reference of GameElement list and cross checks each GameElement for collisions and takes action depending on the type and elasticity of the elements colliding. Also applies gravity, friction and momentum on GameElement objects.

6.2.4 Level Class

This class keeps the reference of a GameElement list which practically means it keeps the reference of the actual specific level.

loadLevel(String): This method loads the GameElement objects from a file with a specific parsing algorithm.

saveLevel(File): This method saves the GameElement objects as string to be parsed by the level class.

6.2.5 GameElement Class

This class itself is abstract and can’t be instantiated. All elements with physical presence on a level extends this class.

x, y, width, height: Position and dimensions of this element’s rectangular area.

verticalSpeed, horizontalSpeed: Velocity of the element on specified axis.

weight, elasticity: Affects the way this object’s collisions are handled (momentum).

flying: If this property is true, this element is not affected by gravity.

smooth: If this property is false, this element will not be physically affected by collisions (But will still be informed in case it collides with another object).

active: This object will be removed from the level by the PhysicsEngine on next timestep if this property is false.

fricted: If this property is false, friction will not be applied on this object.

fixed: If this property is true, this object will not be physically affected by the PhysicsEngine but other objects will still collide with it, meaning that on ly the colliding object will be affected by the collision.

getRectangle(): All objects in the game have rectangular hit boxes. intersects method calls this method to check if there is any collision between two elements.

intersects(GameElement): Checks if the object collides with said another GameElement.

contact(String, GameElement): PhysicsEngine informs the object of the collision and gives the reference of the colliding object and gives its direction (left, right, top, bottom). It is an empty method by default, extending element can override it for element specific actions on contacts.

timestep(double, List<GameElement>): This method is called on every loop of PhysicsEngine timestep call (not to be confused with timestep method inside PhysicsEngine). It is empty by default but some elements might require specific actions to be performed other than physics every timestep, it can be overridden to fulfill that possible element specific requirement.

draw(Graphics, Point): Draws the object on a given Graphics object with a given offset. Should be overridden to make sure all objects on the level look different.

6.2.6 StaticElement Class

These objects are just implementations of GameElements which have their “fixed” property true. However this does not mean that they are stationary in the level. They do not have unpredictable movements like the Player or MoveableObject but door and platform can have a periodic predictable movement if they are triggered.

Triggerable: This is an interface which can be implemented by anything. It can be an element which moves accordingly when triggered or an ordinary object which may end the level when triggered.

Door: This element is a thin tall object which decreases height towards upwards when triggered. It may block the path of the player in-game, player must find a way to trigger it to get through.

Platform: This element is similar to Door but it is more complex. It can be set to move any amount of pixels and its direction and axis can be set. Its timestep function is overridden to check for any non-fixed objects and drag them with it (to ensure that a player standing on top of it during movement does not fall off).

6.2.7 Trigger Class

Trigger: A trigger is an abstract class which is extended by various trigger objects. These objects basically allow the player to activate Triggerable objects to trigger dynamic events within the level such as opening a door, setting a checkpoint, or even end the game.

Lever: A lever is a usable object toggled by the player via the use of interaction key while standing on top of it (Which also means it is not a smooth object).

Button: The button is extremely similar to the lever since it is just a timed version of lever. Depending on the level design, these objects only trigger its Triggerable for a limited amount of time.

Contact Trigger: A simple type of trigger which is activated when a non-static object is in it.

Player Trigger: Same as a contact trigger, except this one is only triggered when a player is in it. Usually used to set checkpoints and end levels.

6.2.8 Player Class

Player class is one of the most important elements in the game. This is the element which represents the main character of the game and it is directly controlled by the user. It has the ability to jump up to a set height and can run horizontally with a set speed.

tool: A tool object which can be used by the user with a mouse click.

useTool(): Receives mouse location from GamePanel and passes it over to the Tool class for execution.

6.2.9 Tool Class

This abstract class enables the user to utilize mechanics other than the regular game mechanics such as running and jumping. This class can be extended in order to give new abilities to the user and mechanics to the game.

BoomerangTool: This tool is more like a control class to enable the user to throw a boomerang which comes back to the player after reaching its maximum range or after any collision. It can activate triggers and push other moveable objects. The player will be able to use this tool on certain levels (depending on the level design) and these levels will possibly require the user to use this tool cleverly to activate triggers remotely or to relocate moveable objects.

Rope: Just like BoomerangTool, this class controls the Hook object it produces when it is used. Hook goes away from the player until the maximum range is reached or until it collides with another object. But unlike Boomerang object, this object pulls the player to its point of collision (if it collides with anything).

7 Conclusion

In this report, it is tried to explain the project and its purposes, functionality of this project, project analysis and design details. As mentioned earlier, Dungeon Escape is a 2D side scrolling-puzzle platform game. The main goal of this game is that challenge the player with difficult puzzles.

Development of project is completed in three main steps which are project analysis, system design and object design. In the project analysis step, Requirement Analysis and Analysis Models are explained in detail. Overview of the project, scenarios, use cases, object models, functional and non-functional requirements are included in the Requirement Analysis part. Together with that, state charts and sequence diagrams are included in dynamic model of the system. We also included how the basic menus of the project look with screen-mock-ups.

In the system design step, design goals, subsystem decomposition, hardware/ software mapping, persistent data management, boundary conditions, access control and security are handled. Also subsystem services and game subsystems are explained in detail in the rest of the design phase.

We tried to handle problems step by step in each project report. Sometimes new ideas came up and some of them were added to the project and its subsystems carefully. We are familiar with code part but we had some problems with the new topics like patterns and diagrams. At the end, we handled these problems with and applied to our project. Our project was shaped with these problems, project reports and feedbacks.

We have learned project development cycle, individual processes with practicing. In addition to that, we have learned how to work within a group project and team working. Also we practiced our Java and object-oriented skills. We are now more familiar with basic principles of object-oriented design. Most importantly we learned the communication between client and the programmer and how should it be handled during the process of the project.

Appendix

```
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Point;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.List;

import javax.imageio.ImageIO;

import com.dungeonescape.tool.BoomerangTool;

public class Boomerang extends GameElement {
	private BoomerangTool thrower;
	private double speed;
	private boolean returning;
	private int range;
	private double angle;
	private BufferedImage image;

	public Boomerang(double x, double y, double angle, double speed,
			BoomerangTool s) {
		super(x, y);
		setFlying(true);
		setFricted(false);
		verticalSpeed = Math.sin(angle) * speed;
		horizontalSpeed = Math.cos(angle) * speed;
		width = 15;
		height = 15;
		returning = false;
		thrower = s;
		range = 200;
		this.angle = angle;
		try {
			image = ImageIO.read(new File("img/boomerang.png"));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public Boomerang() {
		this(0, 0, 0, 0, null);
		setActive(false);
	}

	@Override
	public void contact(int direction, GameElement e) {
		if (e != thrower.getOwner())
			returning = true;
	}

	@Override
	public void timestep(double d, List<GameElement> elementsInWorld)
{
		double distance = Math.hypot(getCenter().x - thrower.getX(),
				getCenter().y - thrower.getY());
		if (distance >= range) {
			returning = true;
		} else if (distance < width && returning) {
			setActive(false);
		}
	}

	@Override
	public boolean intersects(GameElement e) {
		if (returning)
			return false;
		boolean intersects = super.intersects(e);
		return intersects;
	}

	@Override
	public void draw(Graphics g, Point camera) {
		g.drawImage(image, (int) x - camera.x, (int) y - camera.y,
null);
	}

	public double getSpeed() {
		return speed;
	}

	public void setSpeed(double speed) {
		this.speed = speed;
	}

	public BoomerangTool getThrower() {
		return thrower;
	}

	public void setThrower(BoomerangTool thrower) {
		this.thrower = thrower;
	}

	public boolean isReturning() {
		return returning;
	}

	public void setReturning(boolean returning) {
		this.returning = returning;
	}

	public int getRange() {
		return range;
	}

	public void setRange(int range) {
		this.range = range;
	}

	public Point getCenter() {
		return new Point((int) x + width / 2, (int) y + height / 2);
	}

	public double getAngle() {
		return angle;
	}

	public void setAngle(double angle) {
		this.angle = angle;
	}
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Point;

import com.dungeonescape.common.Images;

public class Button extends Trigger {
	public Button(double x, double y) {
		super(x, y);
		setWidth(10);
		setHeight(10);
	}

	@Override
	public void useAction() {
		activate();
	}

	@Override
	public void draw(Graphics g, Point camera) {
		super.draw(g, camera);
		if (isTriggerActive())
			g.drawImage(Images.BUTTON_ON, (int) x - camera.x, (int)
y
					- camera.y, null);
		else
			g.drawImage(Images.BUTTON_OFF, (int) x - camera.x,
(int) y
					- camera.y, null);
	}
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Point;
import java.awt.image.BufferedImage;

import com.dungeonescape.common.Images;

public class ContactTrigger extends Trigger {
	public ContactTrigger(double x, double y, int width, int height) {
		super(x, y);
		setWidth(width);
		setHeight(height);
	}
	@Override
	public void contact(int direction, GameElement e) {
		if (getTriggerable() != e && !(e instanceof StaticElement))
{
			activate();
		}
	}

	@Override
	public void draw(Graphics g, Point camera) {
		super.draw(g, camera);
		BufferedImage img;
		if (isTriggerActive())
			img = Images.CONTACT_ON;
		else
			img = Images.CONTACT_OFF;
		g.drawImage(img, (int) x - camera.x, (int) y - camera.y,
width, height, null);
	}
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.Point;
import java.awt.TexturePaint;
import java.awt.geom.Rectangle2D;
import java.util.List;

import com.dungeonescape.common.Images;

public class Door extends StaticElement implements Triggerable {
	private double doorOpening;
	private int maxHeight;
	private boolean closed;
	private double speed;

	public Door(double x, double y) {
		super(x, y);
		width = 10;
		height = 50;
		doorOpening = height;
		maxHeight = height;
		speed = 8;
		closed = true;
	}

	@Override
	public void timestep(double d, List<GameElement> elementsInWorld)
{
		super.timestep(d, elementsInWorld);
		if (closed) {
			if (doorOpening < maxHeight)
				doorOpening += speed * d;
			if (doorOpening > maxHeight)
				doorOpening = maxHeight;
		} else {
				if (doorOpening > 0)
					doorOpening -= speed * d;
				if (doorOpening < 0)
					doorOpening = 0;
			}

			height = (int) doorOpening;
		}

		@Override
		public void setHeight(int height) {
			maxHeight = height;
		}

		@Override
		public void draw(Graphics g, Point camera) {
			Graphics2D g2 = (Graphics2D) g;
			TexturePaint tp = new TexturePaint(Images.DOOR, new
	Rectangle2D.Double(
					getX() - camera.x, getY() - camera.y
							- (maxHeight - (int) doorOpening),
					Images.DOOR.getWidth(),
	Images.DOOR.getHeight()));
			Paint prevPaint = g2.getPaint();
			g2.setPaint(tp);
			g.fillRect((int) x - camera.x, (int) y - camera.y, width,
	height);
			g2.setPaint(prevPaint);
		}

		public boolean isClosed() {
			return closed;
		}

		public void setClosed(boolean closed) {
			this.closed = closed;
		}

		public void toggleClosed() {
			closed = !closed;
		}

		public double getSpeed() {
			return speed;
		}

		public void setSpeed(double speed) {
			this.speed = speed;
		}

		@Override
		public void trigger(boolean b, Trigger t) {
			if (b) {
				closed = false;
			} else {
				closed = true;
			}
		}
```

```
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Point;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import java.util.List;

import com.dungeonescape.common.ContactConstants;

public abstract class GameElement {
	protected int width, height;
	protected double x, y, verticalSpeed, horizontalSpeed, weight,
elasticity;
	protected boolean flying, smooth, active, fricted, fixed;

	public GameElement(double x, double y) {
		this.x = x;
		this.y = y;
		flying = false;
		smooth = true;
		fricted = true;
		verticalSpeed = 0;
		horizontalSpeed = 0;
		active = true;
		width = 20;
		height = 40;
		weight = 10;
		fixed = false;
		elasticity = 0.5;
	}

	public Rectangle2D getRectangle() {
		return new Rectangle2D.Double(x, y, width, height);
	}

	public Line2D getLine() {
		return null;
	}

	public boolean intersects(GameElement e) {
		if (e.getLine() == null) {
			if (getRectangle().intersects(e.getRectangle())) {
				if (!e.isSmooth()) {
					e.contact(ContactConstants.IN, this);
					return false;
				}
				return true;
			}
			return false;
		} else {
			if (getRectangle().intersectsLine(e.getLine())) {
				if (!e.isSmooth()) {
					e.contact(ContactConstants.IN, this);
					return false;
				}
				return true;
			}
			return false;
		}
	}

	public Point getCenter() {
		return new Point((int) x + width / 2, (int) y + height / 2);
	}

	public void contact(int direction, GameElement e) {
	}

	public void timestep(double d, List<GameElement> elementsInWorld)
{
	}

	public void useAction() {
	}

	public double getX() {
		return (int) x;
	}

	public double getY() {
		return (int) y;
	}

	public void moveX(double xChange) {
		x += xChange;
	}

	public void moveY(double yChange) {
		y += yChange;
	}

	public void draw(Graphics g, Point camera) {
	}

	public double getVerticalSpeed() {
		return verticalSpeed;
	}

	public void setVerticalSpeed(int verticalSpeed) {
		this.verticalSpeed = verticalSpeed;
	}

	public boolean isFlying() {
		return flying;
	}

	public void setFlying(boolean flying) {
		this.flying = flying;
	}

	public boolean isSmooth() {
		return smooth;
	}
```

```
public void setSmooth(boolean smooth) {
      this.smooth = smooth;
}

public boolean isActive() {
      return active;
}

public void setActive(boolean active) {
      this.active = active;
}

public boolean isFricted() {
      return fricted;
}

public void setFricted(boolean fricted) {
      this.fricted = fricted;
}

public void setX(double x) {
      this.x = x;
}

public void setY(double y) {
      this.y = y;
}

public int getWidth() {
      return width;
}

public void setWidth(int width) {
      this.width = width;
}

public int getHeight() {
      return height;
}

public void setHeight(int height) {
      this.height = height;
}

public double getHorizontalSpeed() {
      return horizontalSpeed;
}

public void setHorizontalSpeed(double horizontalSpeed) {
      this.horizontalSpeed = horizontalSpeed;
}

public void setVerticalSpeed(double verticalSpeed) {
      this.verticalSpeed = verticalSpeed;
}

public double getWeight() {
		return weight;
	}

	public void setWeight(double weight) {
		this.weight = weight;
	}

	public boolean isFixed() {
		return fixed;
	}

	public void setFixed(boolean fixed) {
		this.fixed = fixed;
	}

	public double getElasticity() {
		return elasticity;
	}

	public void setElasticity(double elasticity) {
		this.elasticity = elasticity;
	}
}
package com.dungeonescape.element;

import java.awt.BasicStroke;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Stroke;

import com.dungeonescape.tool.Rope;

public class Hook extends GameElement {
	private boolean grappled;
	private Rope source;

	public Hook(double x, double y, double angle, double speed, Rope 
s) {
		super(x, y);
		setFlying(true);
		setSmooth(false);
		grappled = false;
		setFricted(false);
		verticalSpeed = Math.sin(angle) * speed;
		horizontalSpeed = Math.cos(angle) * speed;
		source = s;
		width = 5;
		height = 5;
	}

	public Hook() {
		this(0, 0, 0, 0, null);
		setActive(false);
	}

	@Override
	public boolean intersects(GameElement e) {
```

```
            if (e instanceof StaticElement) {
                  if (super.intersects(e)) {
                        horizontalSpeed = 0;
                        verticalSpeed = 0;
                        grappled = true;
                  }
            }
            return false;
      }

      public void contact(String direction, GameElement e) {
      }

      public void moveX(double xChange) {
            super.moveX(xChange);
            if (Math.hypot(getCenter().x - source.getX(),
                        getCenter().y - source.getY()) >=
source.getReach()
                        && isActive()) {
                  setActive(false);
            }
      }

      public void moveY(double yChange) {
            super.moveY(yChange);
            if (Math.hypot(getCenter().x - source.getX(),
                        getCenter().y - source.getY()) >=
source.getReach()
                        && isActive()) {
                  setActive(false);
            }
      }

      public Point getCenter() {
            return new Point((int) x + width / 2, (int) y + height / 2);
      }

      public void obstruction(String direction, GameElement e) {
            if (e instanceof Obstacle) {
                  horizontalSpeed = 0;
                  verticalSpeed = 0;
                  grappled = true;
            }
      }

      public void draw(Graphics g, Point camera) {
            if (source != null) {
                  Graphics2D g2 = (Graphics2D) g;
                  Stroke s = g2.getStroke();
                  g2.setStroke(new BasicStroke(3));
                  g2.drawLine((int) x + width / 2  - camera.x, (int) y +
height / 2 - camera.y,
                              (int) source.getX() - camera.x, (int)
source.getY() - camera.y);
                  g2.setStroke(s);
            }
            g.fillRect((int) x - camera.x, (int) y - camera.y, width,
height);
      }

      public boolean isGrappled() {
            return grappled;
      }

      public void setGrappled(boolean grappled) {
            this.grappled = grappled;
      }

      public Rope getSource() {
            return source;
      }

      public void setSource(Rope source) {
            this.source = source;
      }
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Point;
import java.util.List;

import com.dungeonescape.common.Images;

public class Lever extends Trigger {
      private boolean leverActive;

      public Lever(double x, double y) {
            super(x, y);
            setWidth(10);
            setHeight(20);
            leverActive = false;
      }

      @Override
      public void timestep(double d, List<GameElement> elementsInWorld)
{
            if (leverActive)
                  activate();
            super.timestep(d, elementsInWorld);
      }

      @Override
      public void useAction() {
            leverActive = !leverActive;
      }

      @Override
      public void draw(Graphics g, Point camera) {
            super.draw(g, camera);
            if (isTriggerActive())
                  g.drawImage(Images.LEVER_ON, (int) x - camera.x,
                              (int) y - camera.y, null);
            else
```

```
                g.drawImage(Images.LEVER_OFF, (int) x - camera.x, (int)
y
                            - camera.y, null);
      }

      public boolean isLeverActive() {
            return leverActive;
      }

      public void setLeverActive(boolean leverActive) {
            this.leverActive = leverActive;
      }
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Point;
import java.awt.geom.Line2D;

public class LinearTrigger extends ContactTrigger {
      private int xEnd, yEnd;

      public LinearTrigger(double x, double y, int xEnd, int yEnd) {
            super(x, y, 0, 0);
            this.xEnd = xEnd;
            this.yEnd = yEnd;
      }

      @Override
      public Line2D getLine() {
            return new Line2D.Double(x, y, xEnd, yEnd);
      }

      @Override
      public void draw(Graphics g, Point camera) {
            super.draw(g, camera);
            g.drawLine((int)x - camera.x, (int)y - camera.y, xEnd -
camera.x, yEnd - camera.y);
      }

      public int getxEnd() {
            return xEnd;
      }

      public void setxEnd(int xEnd) {
            this.xEnd = xEnd;
      }

      public int getyEnd() {
            return yEnd;
      }

      public void setyEnd(int yEnd) {
            this.yEnd = yEnd;
      }
}
package com.dungeonescape.element;
```

```
import java.awt.Graphics;
import java.awt.Point;

import com.dungeonescape.common.Images;

public class MoveableObject extends GameElement {
      final int WIDTH = 20;
      final int HEIGHT = 40;

      public MoveableObject(int x, int y) {
            super(x, y);
      }

      public void draw(Graphics g, Point camera) {
            g.drawImage(Images.MOVEABLE_OBJECT, (int) x - camera.x,
(int) y
                        - camera.y, null);
      }

      public void obstruction(String direction, GameElement e) {
      }
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.Point;
import java.awt.TexturePaint;
import java.awt.geom.Rectangle2D;

import com.dungeonescape.common.Images;

public class Obstacle extends StaticElement {
      public Obstacle(int x, int y, int w, int h) {
            super(x, y);
            width = w;
            height = h;
      }

      public void draw(Graphics g, Point camera) {
            Graphics2D g2 = (Graphics2D) g;
            TexturePaint tp = new TexturePaint(
                        Images.OBSTACLE,
                        new Rectangle2D.Double(getX() - camera.x, getY()
-camera.y,
                                    Images.OBSTACLE.getWidth(),
Images.OBSTACLE.getHeight()));
            Paint prevPaint = g2.getPaint();
            g2.setPaint(tp);
            g2.fillRoundRect((int) x - camera.x, (int) y - camera.y,
width, height, 5, 5);
            g2.setPaint(prevPaint);
      }
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.Point;
import java.awt.TexturePaint;
import java.awt.geom.Rectangle2D;
import java.util.List;

import com.dungeonescape.common.Images;

public class Platform extends StaticElement implements Triggerable {
	private double travel;
	private int maxTravel;
	private boolean activated;
	private double speed;
	private boolean verticalTravel;
	private double origin;

	public Platform(int x, int y, int width, int height, int
maxTravel,
			boolean verticalTravel) {
		super(x, y);
		this.width = width;
		this.height = height;
		speed = 8;
		this.maxTravel = maxTravel;
		activated = false;
		travel = 0;
		this.verticalTravel = verticalTravel;
		if (verticalTravel)
			origin = y;
		else
			origin = x;
	}

	@Override
	public void draw(Graphics g, Point camera) {
		Graphics2D g2 = (Graphics2D) g;
		TexturePaint tp = new TexturePaint(
				Images.PLATFORM,
				new Rectangle2D.Double(getX() - camera.x, getY()
- camera.y,
						Images.PLATFORM.getWidth(),
Images.PLATFORM.getHeight()));
		Paint prevPaint = g2.getPaint();
		g2.setPaint(tp);
		g2.fillRoundRect((int) x - camera.x, (int) y - camera.y,
width, height,
				5, 5);
		g2.setPaint(prevPaint);
	}

	@Override
	public void timestep(double d, List<GameElement> elementsInWorld)
{
		super.timestep(d, elementsInWorld);
		double diff = 0;
		if (activated) {
			if (maxTravel > 0) {
```

```
                if (travel < maxTravel)
                    diff = speed * d;
            } else {
                if (travel > maxTravel)
                    diff = -speed * d;
            }
        } else {
            if (travel > 0)
                diff = -speed * d;
            else if (travel < 0)
                diff = speed * d;
        }
        travel += diff;
        if (verticalTravel) {
            moveY(diff);
        } else {
            moveX(diff);
        }
        moveY(-1);
        for (GameElement e : elementsInWorld) {
            if (this.intersects(e) && !e.isFixed() && e.isSmooth())
{
                if (verticalTravel)
                    e.moveY(diff);
                else
                    e.moveX(diff);
            }
        }
        moveY(1);
    }

    @Override
    public void trigger(boolean b, Trigger t) {
        if (b) {
            activated = true;
        } else {
            activated = false;
        }
    }

    public double getTravel() {
        return travel;
    }

    public void setTravel(double travel) {
        this.travel = travel;
    }

    public int getMaxTravel() {
        return maxTravel;
    }

    public void setMaxTravel(int maxTravel) {
        this.maxTravel = maxTravel;
    }

    public double getSpeed() {
        return speed;
```

```
	}

	public void setSpeed(double speed) {
		this.speed = speed;
	}

	public boolean isVerticalTravel() {
		return verticalTravel;
	}

	public void setVerticalTravel(boolean verticalTravel) {
		this.verticalTravel = verticalTravel;
	}

	public double getOrigin() {
		return origin;
	}

	public void setOrigin(double origin) {
		this.origin = origin;
	}

	public boolean isActivated() {
		return activated;
	}

	public void setActivated(boolean activated) {
		this.activated = activated;
	}
}
package com.dungeonescape.element;

import java.awt.Graphics;
import java.awt.Point;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.List;

import javax.imageio.ImageIO;

import com.dungeonescape.common.CommonMethods;
import com.dungeonescape.common.ContactConstants;
import com.dungeonescape.tool.Tool;

public class Player extends GameElement {
	private final int ANIMATION = 5;

	private boolean direction;
	private Tool tool;
	private BufferedImage[] images, imagesInverted;
	private int stage;
	private boolean ground, jumping, left, right, jump, using;
	private double verticalacc, horizontalacc, jumpHeight;

	public Player(double x, double y) {
		super(x, y);
		direction = true;
```

```
            jumpHeight = 10;
            images = new BufferedImage[10];
            for (int i = 0; i < images.length; i++) {
                  try {
                        images[i] = ImageIO.read(new File("img/player/" +
i + ".png"));
                  } catch (IOException e) {
                        e.printStackTrace();
                  }
            }
            imagesInverted = new BufferedImage[images.length];
            for (int i = 0; i < imagesInverted.length; i++) {
                  imagesInverted[i] =
CommonMethods.horizontalflip(images[i]);
            }
            stage = 0;
            verticalacc = 4;
            horizontalacc = 2;
            elasticity = 0;
            ground = false;
            jumping = false;
            jump = false;
            left = false;
            right = false;
            using = false;
      }

      @Override
      public void contact(int direction, GameElement e) {
            if (direction == ContactConstants.BOTTOM) {
                  ground = true;
            } else if (direction == ContactConstants.TOP) {
                  jumping = false;
            }
      }

      @Override
      public void timestep(double d, List<GameElement> elementsInWorld)
{
            if (using) {
                  for (GameElement e : elementsInWorld) {
                        if (!e.isSmooth()) {
                              if
(e.getRectangle().intersects(getRectangle())) {
                                    e.useAction();
                                    using = false;
                              }
                        }
                  }
            }
            if (right) {
                  if (horizontalSpeed < horizontalacc * 10)
                        horizontalSpeed += horizontalacc * d;
            } else if (horizontalSpeed > 0) {
                  horizontalSpeed -= horizontalacc * d;
            }
            if (left) {
                  if (horizontalSpeed > horizontalacc * -10)
            horizontalSpeed -= horizontalacc * d;
    } else if (horizontalSpeed < 0) {
        horizontalSpeed += horizontalacc * d;
    }
    if (jump) {
        if (ground) {
            ground = false;
            jumping = true;
        }

        if (jumping) {
            if (verticalSpeed > -jumpHeight)
                verticalSpeed -= verticalacc * d;
            else {
                jumping = false;
            }
        }
    }
    if (ground) {
        verticalSpeed = 0;
    }
    ground = false;

    if (tool != null)
        tool.timestep(d);

    if ((left || right) && !(left && right)) {
        animate();
    }
}

public void left(boolean b) {
    left = b;
}

public void right(boolean b) {
    right = b;
}

public void jump(boolean b) {
    if (b)
        jump = true;
    else {
        jump = false;
        jumping = false;
    }
}

public GameElement useTool(int x, int y) {
    if (tool != null)
        return tool.use(x, y);
    return null;
}

public void moveX(double xChange) {
    x += xChange;
    if (tool != null) {
        tool.setX(tool.getX() + xChange);
```

```
		}
	}

	public void moveY(double yChange) {
		y += yChange;
		if (tool != null) {
			tool.setY(tool.getY() + yChange);
		}
	}

	public void setDirection(boolean b) {
		direction = b;
	}

	public void draw(Graphics g, Point camera) {
		if (direction) {
			if (!ground)
				g.drawImage(images[9], (int) x - 10 - camera.x,
(int) y
						- camera.y, null);
			else if (Math.abs(horizontalSpeed) >= 0.1)
				g.drawImage(images[stage / ANIMATION + 1], (int)
x - 10 - camera.x,
						(int) y - camera.y, null);
			else
				g.drawImage(images[0], (int) x - 10 - camera.x,
(int) y
						- camera.y, null);
		} else {
			if (!ground)
				g.drawImage(imagesInverted[9], (int) x - 10 -
camera.x, (int) y
						- camera.y, null);
			else if (Math.abs(horizontalSpeed) >= 0.1)
				g.drawImage(imagesInverted[stage / ANIMATION +
1], (int) x - 10
						- camera.x, (int) y - camera.y, null);
			else
				g.drawImage(imagesInverted[0], (int) x - 10 -
camera.x, (int) y
						- camera.y, null);
		}
	}

	public void animate() {
		stage++;
		if (stage / ANIMATION >= 8)
			stage = 0;
	}

	public Tool getTool() {
		return tool;
	}

	public void setTool(Tool tool) {
		this.tool = tool;
		tool.setOwner(this);
		tool.setX(x + width / 2);
		tool.setY(y + height / 2);
	}

	public boolean isGround() {
		return ground;
	}

	public void setGround(boolean ground) {
		this.ground = ground;
	}

	public boolean isJumping() {
		return jumping;
	}

	public void setJumping(boolean jumping) {
		this.jumping = jumping;
	}

	public double getVerticalacc() {
		return verticalacc;
	}

	public void setVerticalacc(double verticalacc) {
		this.verticalacc = verticalacc;
	}

	public double getHorizontalacc() {
		return horizontalacc;
	}

	public void setHorizontalacc(double horizontalacc) {
		this.horizontalacc = horizontalacc;
	}

	public double getJumpHeight() {
		return jumpHeight;
	}

	public void setJumpHeight(int jumpHeight) {
		this.jumpHeight = jumpHeight;
	}

	public boolean isUsing() {
		return using;
	}

	public void use() {
		this.using = true;
	}

	public void setX(double x) {
		super.setX(x);
		if (tool != null)
			tool.setX(getCenter().x);
	}

	public void setY(double y) {
			super.setY(y);
			if (tool != null)
				tool.setY(getCenter().y);
		}
	}
package com.dungeonescape.element;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Point;

import com.dungeonescape.common.TriggerConstants;
import com.dungeonescape.game.Game;

public class PlayerTrigger extends ContactTrigger {
	public PlayerTrigger(double x, double y, int width, int height) {
		super(x, y, width, height);
	}

	@Override
	public void contact(int direction, GameElement e) {
		if (e instanceof Player)
			super.contact(direction, e);
	}

	@Override
	public void draw(Graphics g, Point camera) {
		Font f = g.getFont();
		Color c = g.getColor();
		g.setColor(Color.white);
		g.setFont(new Font("Calibri", Font.PLAIN, 16));
		int type = 0;
		if(getTriggerable() instanceof Game.CheckPoint)
			type = TriggerConstants.CHECKPOINT;
		else if (getTriggerable() instanceof Game.GameEnder)
			type = TriggerConstants.ENDER;
		else if (getTriggerable() instanceof SavedTriggerable)
			type =
((SavedTriggerable)getTriggerable()).getGameElementNo();

		if (type == TriggerConstants.CHECKPOINT) {
			g.drawString("Checkpoint", (int) x - camera.x, (int) y
- camera.y + 15);
			g.setColor(new Color(255, 255, 0, 50));
		} else if (type == TriggerConstants.ENDER) {
			g.drawString("Goal", (int) x - camera.x, (int) y -
camera.y + 15);
			g.setColor(new Color(0, 255, 0, 50));
		} else {
			g.setColor(new Color(255, 255, 255, 50));
		}
		g.fillRoundRect((int) x - camera.x, (int) y - camera.y,
width, height,
				5, 5);
		g.setColor(c);
		g.setFont(f);
	}
```

```
}
package com.dungeonescape.element;

public class SavedTriggerable implements Triggerable {
      private int gameElementNo;

      public SavedTriggerable(int no) {
            gameElementNo = no;
      }

      @Override
      public void trigger(boolean b, Trigger t) {
      }

      public int getGameElementNo() {
            return gameElementNo;
      }

      public void setGameElementNo(int gameElementNo) {
            this.gameElementNo = gameElementNo;
      }
}
package com.dungeonescape.element;

public abstract class StaticElement extends GameElement {
      public StaticElement(double x, double y) {
            super(x, y);
            setFlying(true);
            setFixed(true);
      }

      public void obstruction(String direction, GameElement e) {
      }
}
package com.dungeonescape.element;

import java.awt.AlphaComposite;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Stroke;
import java.awt.geom.Line2D;
import java.util.List;

public class Trigger extends StaticElement implements Triggerable {
      private Triggerable triggerable;
      private double triggerActive;
      private int triggerDuration;
      private int delay;
      private double remainingDelay;

      public Trigger(double x, double y) {
            super(x, y);
            smooth = false;
            triggerActive = 0;
            triggerDuration = 1;
            delay = 0;
            remainingDelay = 0;
      }

      @Override
      public void timestep(double d, List<GameElement> elementsInWorld)
{
            super.timestep(d, elementsInWorld);
            if (triggerable != null) {
                  if (triggerActive > 0) {
                        if (remainingDelay <= 0) {
                              triggerable.trigger(true, this);
                              triggerActive -= d;
                        } else {
                              remainingDelay -= d;
                        }
                  } else {
                        triggerable.trigger(false, this);
                        remainingDelay = delay;
                  }
            }
      }

      public void activate() {
            triggerActive = triggerDuration;
      }

      public void trigger(boolean b, Trigger t) {
            if (b)
                  activate();
      }

      public Triggerable getTriggerable() {
            return triggerable;
      }

      public void setTriggerable(Triggerable triggerable) {
            this.triggerable = triggerable;
      }

      public int getTriggerDuration() {
            return triggerDuration;
      }

      public void setTriggerDuration(int triggerDuration) {
            this.triggerDuration = triggerDuration;
      }

      public int getDelay() {
            return delay;
      }

      public void setDelay(int delay) {
            this.delay = delay;
            remainingDelay = delay;
      }

      public boolean hasTriggerable() {
```

```
		return triggerable != null;
	}

	public boolean isTriggerActive() {
		return triggerActive > 0;
	}

	@Override
	public void draw(Graphics g, Point camera) {
		if (getTriggerable() instanceof GameElement) {
			Graphics2D g2 = ((Graphics2D) g);
			Color prevColor = g2.getColor();
			Stroke prevStroke = g2.getStroke();
			Point c = getCenter();
			c.x -= camera.x;
			c.y -= camera.y;
			GameElement triggerable = ((GameElement)
getTriggerable());
			while (triggerable instanceof Trigger) {
				if (((Trigger) triggerable).getTriggerable()
instanceof GameElement)
					triggerable = (GameElement) ((Trigger)
triggerable)
							.getTriggerable();
				else
					break;
			}
			Point t = triggerable.getCenter();
			t.x -= camera.x;
			t.y -= camera.y;
			Line2D connection = new Line2D.Double(c, t);
			g2.setColor(Color.green);

	g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER
,
					0.2f));
			g2.setStroke(new BasicStroke(5, BasicStroke.CAP_BUTT,
					BasicStroke.JOIN_BEVEL));
			g2.draw(connection);
			g2.setColor(Color.yellow);

	g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER
,
					1));
			float dash1[] = {10.0f, 5.0f, 3.0f};
			g2.setStroke(new BasicStroke(1.0f,
BasicStroke.CAP_BUTT,
					BasicStroke.JOIN_MITER, 10.0f, dash1,
0.0f));
			g2.draw(connection);
			g2.setColor(prevColor);
			g2.setStroke(prevStroke);
		}
	}
}
package com.dungeonescape.element;

public interface Triggerable {
        void trigger(boolean b, Trigger t);
}
```

```
package com.dungeonescape.common;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Point;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;

public class CommonMethods {
	public static BufferedImage horizontalflip(BufferedImage img) {
		int w = img.getWidth();
		int h = img.getHeight();
		BufferedImage dimg = new BufferedImage(w, h, img.getType());
		Graphics2D g = dimg.createGraphics();
		g.drawImage(img, 0, 0, w, h, w, 0, 0, h, null);
		g.dispose();
		return dimg;
	}

	private static class SoundThread extends Thread {
		String file;

		public SoundThread(String file) {
			this.file = file;
		}

		public void run() {
			File yourFile = new File(file);
			AudioInputStream stream = null;
			AudioFormat format;
			DataLine.Info info;
			Clip clip = null;

			try {
				stream =
AudioSystem.getAudioInputStream(yourFile);
			} catch (UnsupportedAudioFileException e) {
				e.printStackTrace();
			} catch (IOException e) {
				e.printStackTrace();
			}
			format = stream.getFormat();
			info = new DataLine.Info(Clip.class, format);
			try {
				clip = (Clip) AudioSystem.getLine(info);
			} catch (LineUnavailableException e) {
				e.printStackTrace();
```

```
                }
                try {
                      clip.open(stream);
                } catch (LineUnavailableException e) {
                      e.printStackTrace();
                } catch (IOException e) {
                      e.printStackTrace();
                }
                clip.start();
          }
    }

    public static void playSound(String file) {
          (new SoundThread(file)).start();
    }

    public static void textureDraw(Image image, Graphics g, Dimension
size,
                Point offset) {
          for (int x = 0; x < size.getWidth(); x +=
image.getWidth(null)) {
                for (int y = 0; y < size.getHeight(); y +=
image.getHeight(null)) {
                      g.drawImage(image, x, y, null);
                }
          }
    }

    public static BufferedImage convert(BufferedImage image) {
          BufferedImage newImage = new BufferedImage(image.getWidth(),
                      image.getHeight(),
BufferedImage.TYPE_USHORT_555_RGB);
          Graphics2D g = newImage.createGraphics();
          g.drawImage(image, 0, 0, null);
          g.dispose();
          return newImage;
    }
}
package com.dungeonescape.common;

public class ContactConstants {
    public static final int TOP = 0;
    public static final int BOTTOM = 1;
    public static final int LEFT = 2;
    public static final int RIGHT = 3;
    public static final int IN = 4;
}
package com.dungeonescape.common;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;

public class Images {
    public static BufferedImage BUTTON_ON;
    public static BufferedImage BUTTON_OFF;
```

```
	public static BufferedImage LEVER_ON;
	public static BufferedImage LEVER_OFF;
	public static BufferedImage OBSTACLE;
	public static BufferedImage DOOR;
	public static BufferedImage MOVEABLE_OBJECT;
	public static BufferedImage PLATFORM;
	public static BufferedImage CONTACT_OFF;
	public static BufferedImage CONTACT_ON;

	public static void loadImages() {
		try {
			BUTTON_OFF = ImageIO.read(new
File("img/buttonoff.png"));
			BUTTON_ON = ImageIO.read(new File("img/buttonon.png"));
			LEVER_ON = ImageIO.read(new File("img/leveroff.png"));
			LEVER_OFF = ImageIO.read(new File("img/leveron.png"));
			OBSTACLE = ImageIO.read(new File("img/obstacle.png"));
			DOOR = ImageIO.read(new File("img/door.png"));
			MOVEABLE_OBJECT = ImageIO.read(new
File("img/moveableobject.png"));
			PLATFORM = ImageIO.read(new File("img/platform.png"));
			CONTACT_OFF = ImageIO.read(new
File("img/contactoff.png"));
			CONTACT_ON = ImageIO.read(new
File("img/contacton.png"));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
package com.dungeonescape.common;

public class ToolConstants {
	public static final int NONE = 0;
	public static final int BOOMERANG = 1;
	public static final int ROPE = 2;
}
package com.dungeonescape.common;

public class TriggerConstants {
	public static final int CHECKPOINT = -1;
	public static final int ENDER = -2;
}
```

```java
package com.dungeonescape.game;

import java.awt.Graphics;
import java.awt.Point;
import java.io.File;
import java.util.List;

import com.dungeonescape.common.ToolConstants;
import com.dungeonescape.common.TriggerConstants;
import com.dungeonescape.element.GameElement;
import com.dungeonescape.element.Player;
import com.dungeonescape.element.SavedTriggerable;
import com.dungeonescape.element.Trigger;
import com.dungeonescape.element.Triggerable;
import com.dungeonescape.gameio.GamePanel;
import com.dungeonescape.tool.BoomerangTool;
import com.dungeonescape.tool.Rope;

public class Game {
      private GamePanel panel;
      private Level level;
      private PhysicsEngine physicsEngine;
      private Player player;
      private double gravity, friction;
      private boolean stopped;
      private Thread gameThread;
      private GameEnder gameEnder;
      private CheckPoint checkPoint;

      public Game(GamePanel panel) {
            this.panel = panel;
            physicsEngine = new PhysicsEngine();
            setLevel(new Level());
            loadLevel(null);
            gravity = 2;
            friction = 0.5;
            stopped = true;
            setGameEnder(new GameEnder());
            setCheckPoint(new CheckPoint());
      }

      public void start() {
            stopped = false;
            gameThread = new Thread() {
                  public void run() {
                        gameLoop();
                  }
            };
            gameThread.start();
      }

      public void stop() {
            stopped = true;
            if (gameThread != null) {
                  try {
                        gameThread.join();
                  } catch (InterruptedException e) {
                        e.printStackTrace();
```

```
			}
		}
	}

	public void setGravity(int g) {
		gravity = g;
	}

	public void paint(Graphics g, Point camera) {
		for (GameElement e : level.getElements()) {
			e.draw(g, camera);
		}
		player.draw(g, camera);
	}

	public void setPlayer(Player p) {
		level.removeElement(player);
		level.addElement(p);
		player = p;
	}

	public Point getPlayerPosition() {
		return player.getCenter();
	}

	private void gameLoop() {
		final double TARGET_FPS = 60.0;
		final double OPTIMAL_TIME = 1000000000.0 / TARGET_FPS;
		long previous = System.nanoTime();
		long now;
		long elapsed;
		double accumulator = 0.0;

		while (!stopped) {
			now = System.nanoTime();
			elapsed = (now - previous);
			previous = now;
			accumulator += (double) elapsed;

			if (accumulator >= OPTIMAL_TIME) {
				physicsEngine.timestep(1, level.getElements(),
gravity,
						friction);
				if (player.getY() > level.getFallHeight())
					gameEnder.trigger(true, null);
				accumulator -= OPTIMAL_TIME;
				panel.updateUI();
			} else if (accumulator < OPTIMAL_TIME * 0.5) {
				try {
					Thread.sleep(1);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}
	}

	public boolean isStopped() {
		return stopped;
	}

	public void right(boolean b) {
		player.right(b);
	}

	public void left(boolean b) {
		player.left(b);
	}

	public void jump(boolean b) {
		player.jump(b);
	}

	public void use() {
		player.use();
	}

	public void useTool(int x, int y) {
		GameElement e = player.useTool(x, y);
		if (e != null)
			level.addElement(e);
	}

	public Player getPlayer() {
		return player;
	}

	public PhysicsEngine getEngine() {
		return physicsEngine;
	}

	public void setEngine(PhysicsEngine engine) {
		this.physicsEngine = engine;
	}

	public Level getLevel() {
		return level;
	}

	public void setLevel(Level level) {
		this.level = level;
	}

	public void loadLevel(File file) {
		if (file != null)
			level.loadLevel(file);
		List<GameElement> elements = level.getElements();
		for (GameElement e : elements) {
			if (e instanceof Trigger) {
				Trigger t = (Trigger) e;
				if (t.getTriggerable() instanceof
SavedTriggerable) {
					SavedTriggerable st = (SavedTriggerable)
t.getTriggerable();
					if (st.getGameElementNo() >= 0)
```

```
                                  t.setTriggerable((Triggerable)
elements.get(st
                                              .getGameElementNo()));
                            else if (st.getGameElementNo() ==
TriggerConstants.CHECKPOINT)
                                  t.setTriggerable(checkPoint);
                            else if (st.getGameElementNo() ==
TriggerConstants.ENDER)
                                  t.setTriggerable(gameEnder);
                      }
                 }
           }
           player = new Player(level.getSpawnPoint().x,
level.getSpawnPoint().y);
           if (level.getTool() == ToolConstants.BOOMERANG)
                 player.setTool(new BoomerangTool());
           else if (level.getTool() == ToolConstants.ROPE)
                 player.setTool(new Rope());
           level.addElement(player);
           panel.showTip(level.getTip());
     }

     public GameEnder getGameEnder() {
           return gameEnder;
     }

     public void setGameEnder(GameEnder gameEnder) {
           this.gameEnder = gameEnder;
     }

     public CheckPoint getCheckPoint() {
           return checkPoint;
     }

     public void setCheckPoint(CheckPoint checkPoint) {
           this.checkPoint = checkPoint;
     }

     public void reload() {
           level.reload(true);
           loadLevel(null);
     }

     public class GameEnder implements Triggerable {
           @Override
           public void trigger(boolean b, Trigger t) {
                 if (b) {
                      if (t != null) {
                            panel.end(true);
                      } else {
                            panel.end(false);
                      }
                 }
           }
     }

     public class CheckPoint implements Triggerable {
           @Override
```

```
			public void trigger(boolean b, Trigger t) {
				if (b)
					level.setSpawnPoint(new Point((int) t.getX(),
(int) t.getY()));
			}
		}
}
package com.dungeonescape.game;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.Point;
import java.awt.TexturePaint;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.List;

import javax.imageio.ImageIO;

import com.dungeonescape.common.CommonMethods;
import com.dungeonescape.common.ToolConstants;
import com.dungeonescape.common.TriggerConstants;
import com.dungeonescape.element.Button;
import com.dungeonescape.element.ContactTrigger;
import com.dungeonescape.element.Door;
import com.dungeonescape.element.GameElement;
import com.dungeonescape.element.Lever;
import com.dungeonescape.element.LinearTrigger;
import com.dungeonescape.element.MoveableObject;
import com.dungeonescape.element.Obstacle;
import com.dungeonescape.element.Platform;
import com.dungeonescape.element.PlayerTrigger;
import com.dungeonescape.element.SavedTriggerable;
import com.dungeonescape.element.Trigger;
import com.dungeonescape.element.Triggerable;
import com.dungeonescape.game.Game.CheckPoint;
import com.dungeonescape.game.Game.GameEnder;

public class Level {
	private List<GameElement> elements;
	private Point spawnPoint;
	private int fallHeight, tool;
	private BufferedImage background;
	private String backgroundUrl, tip;
	private File file;

	public Level() {
		elements = new ArrayList<GameElement>();
		spawnPoint = new Point(0, 0);
		fallHeight = 1000;
		tool = ToolConstants.NONE;
		tip = "";
		setBackgroundUrl("back.png");
	}

	public List<GameElement> getElements() {
		return elements;
	}

	public void setElements(List<GameElement> elements) {
		this.elements = elements;
	}

	public void addElement(GameElement e) {
		elements.add(e);
	}

	public void removeElement(GameElement e) {
		elements.remove(e);
	}

	public Point getSpawnPoint() {
		return spawnPoint;
	}

	public void setSpawnPoint(Point spawnPoint) {
		this.spawnPoint = spawnPoint;
	}

	public int getFallHeight() {
		return fallHeight;
	}

	public void setFallHeight(int fallHeight) {
		this.fallHeight = fallHeight;
	}

	public int getTool() {
		return tool;
	}

	public void setTool(int tool) {
		this.tool = tool;
	}

	public String getTip() {
		return tip;
	}

	public void setTip(String tip) {
		this.tip = tip;
	}

	public String getBackgroundUrl() {
		return backgroundUrl;
	}
```

```
      public void setBackgroundUrl(String backgroundUrl) {
            if (!backgroundUrl.isEmpty()) {
                  this.backgroundUrl = backgroundUrl;
                  try {
                        background =
CommonMethods.convert(ImageIO.read(new File("img/"
                                    + backgroundUrl)));
                  } catch (IOException e) {
                        e.printStackTrace();
                  }
            }
      }

      public void paintBackground(Graphics2D g, Point camera, Dimension
size) {
            TexturePaint tp = new TexturePaint(background, new
Rectangle2D.Double(
                        -camera.x / 3, -camera.y / 3,
background.getWidth(),
                        background.getHeight()));
            Paint prevPaint = g.getPaint();
            g.setPaint(tp);
            g.fillRect(0, 0, (int) size.getWidth(), (int)
size.getHeight());
            g.setPaint(prevPaint);
            Color prevColor = g.getColor();
            g.setColor(new Color(0, 0, 0, 125));
            g.fillRect(0, 0, (int) size.getWidth(), (int)
size.getHeight());
            g.setColor(prevColor);
      }

      public File getFile() {
            return file;
      }

      public void saveLevel(File file) {
            try {
                  BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(
                              new FileOutputStream(file)));
                  String toolString = "none";
                  if (tool == ToolConstants.BOOMERANG)
                        toolString = "boomerang";
                  else if (tool == ToolConstants.ROPE)
                        toolString = "rope";
                  bw.write(fallHeight + "");
                  bw.newLine();
                  bw.write(spawnPoint.x + " " + spawnPoint.y);
                  bw.newLine();
                  bw.write(toolString);
                  bw.newLine();
                  bw.write(tip);
                  bw.newLine();
                  bw.write(backgroundUrl);
                  bw.newLine();
                  bw.newLine();
```

```
                 for (GameElement e : elements) {
                       if (e instanceof Obstacle) {
                            bw.write("Obstacle " + (int) e.getX() + " "
                                       + (int) e.getY() + " " +
e.getWidth() + " "
                                       + e.getHeight());
                            bw.newLine();
                       } else if (e instanceof MoveableObject) {
                            bw.write("MoveableObject " + (int) e.getX()
+ " "
                                       + (int) e.getY());
                            bw.newLine();
                       } else if (e instanceof Door) {
                            bw.write("Door " + (int) e.getX() + " " +
(int) e.getY());
                            bw.newLine();
                       } else if (e instanceof Platform) {
                            int vertical = ((Platform)
e).isVerticalTravel() ? 1 : 0;
                            bw.write("Platform " + (int) e.getX() + " "
                                       + (int) e.getY() + " " +
e.getWidth() + " "
                                       + e.getHeight() + " "
                                       + ((Platform) e).getMaxTravel()
+ " " + vertical);
                            bw.newLine();
                       } else if (e instanceof Trigger) {
                            Trigger t = (Trigger) e;
                            if (t instanceof ContactTrigger
                                       && !(t instanceof
PlayerTrigger)) {
                                  bw.write("ContactTrigger " + (int)
e.getX() + " "
                                             + (int) e.getY() + " " +
e.getWidth() + " "
                                             + e.getHeight());
                            } else if (t instanceof Button) {
                                  bw.write("Button " + (int) e.getX() +
" "
                                             + (int) e.getY());
                            } else if (t instanceof Lever) {
                                  bw.write("Lever " + (int) e.getX() + "
"
                                             + (int) e.getY());
                            } else if (t instanceof LinearTrigger) {
                                  LinearTrigger lt = (LinearTrigger) t;
                                  bw.write("LinearTrigger " + (int)
lt.getX() + " "
                                             + (int) lt.getY() + " " +
lt.getxEnd() + " "
                                             + lt.getyEnd());
                            } else if (t instanceof PlayerTrigger) {
                                  bw.write("PlayerTrigger " + (int)
e.getX() + " "
                                             + (int) e.getY() + " " +
e.getWidth() + " "
                                             + e.getHeight());
                            } else {
```

```
                                        bw.write("Trigger " + (int) e.getX() +
" "
                                                + (int) e.getY());
                                }
                                bw.write(" " + t.getTriggerDuration() + " "
+ t.getDelay()
                                        + " ");
                                Triggerable tbl = t.getTriggerable();
                                if (tbl instanceof SavedTriggerable) {
                                        int tid = ((SavedTriggerable)
tbl).getGameElementNo();
                                        if (tid >= 0)
                                                bw.write(tid);
                                        else if (tid ==
TriggerConstants.CHECKPOINT)
                                                bw.write("checkpoint");
                                        else if (tid ==
TriggerConstants.ENDER)
                                                bw.write("ender");
                                } else {
                                        int index = elements.indexOf(tbl);
                                        if (index >= 0)
                                                bw.write(index + "");
                                        else if (tbl instanceof GameEnder)
                                                bw.write("ender");
                                        else if (tbl instanceof CheckPoint)
                                                bw.write("checkpoint");
                                }
                                bw.newLine();
                        }
                }
                bw.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
}

public void connectSavedTriggers() {
        for (GameElement e : elements) {
                if (e instanceof Trigger) {
                        Trigger t = (Trigger) e;
                        if (t.getTriggerable() instanceof
SavedTriggerable) {
                                SavedTriggerable st = (SavedTriggerable)
t.getTriggerable();
                                if (st.getGameElementNo() >= 0)
                                        t.setTriggerable((Triggerable)
elements.get(st
                                                .getGameElementNo()));
                        }
                }
        }
}

public void reload(boolean preserveCheckpoint) {
        Point tmp = spawnPoint;
        loadLevel(file);
        if (preserveCheckpoint) {
                spawnPoint = tmp;
            }
        }

        public void loadLevel(File file) {
            this.file = file;
            try {
                elements.clear();
                FileReader fr = new FileReader(file);
                BufferedReader br = new BufferedReader(fr);
                String line;
                fallHeight = readFallHeight(br.readLine());
                spawnPoint = readSpawnPoint(br.readLine());
                tool = readTool(br.readLine());
                tip = br.readLine();
                setBackgroundUrl(br.readLine());

                while ((line = br.readLine()) != null) {
                    if (line.length() > 6) {
                        int splitter = line.indexOf(' ');
                        String type = line.substring(0, splitter);
                        String argsString =
line.substring(splitter, line.length());
                        List<Integer> args = readArgs(argsString);
                        if (type.equalsIgnoreCase("Obstacle")) {
                            Obstacle e = new Obstacle(args.get(0),
args.get(1),
                                    args.get(2), args.get(3));
                            elements.add(e);
                        } else if
(type.equalsIgnoreCase("MoveableObject")) {
                            elements.add(new
MoveableObject(args.get(0), args
                                    .get(1)));
                        } else if (type.equalsIgnoreCase("Door")) {
                            elements.add(new Door(args.get(0),
args.get(1)));
                        } else if
(type.equalsIgnoreCase("Platform")) {
                            boolean verticalTravel = args.get(5)
!= 0;
                            elements.add(new Platform(args.get(0),
args.get(1),
                                    args.get(2), args.get(3),
args.get(4),
                                    verticalTravel));
                        } else if (type.equalsIgnoreCase("Trigger")
                                ||
type.equalsIgnoreCase("ContactTrigger")
                                ||
type.equalsIgnoreCase("Button")
                                ||
type.equalsIgnoreCase("Lever")
                                ||
type.equalsIgnoreCase("PlayerTrigger")
                                ||
type.equalsIgnoreCase("LinearTrigger")) {
                            Trigger t = null;
```

```
                                int curArg = 0;
                                if (type.equalsIgnoreCase("Trigger"))
{

                                } else if
(type.equalsIgnoreCase("ContactTrigger")) {
                                     t = new
ContactTrigger(args.get(0), args.get(1),
                                               args.get(2),
args.get(3));
                                     curArg = 4;
                                } else if
(type.equalsIgnoreCase("Button")) {
                                     t = new Button(args.get(0),
args.get(1));
                                     curArg = 2;
                                } else if
(type.equalsIgnoreCase("Lever")) {
                                     t = new Lever(args.get(0),
args.get(1));
                                     curArg = 2;
                                } else if
(type.equalsIgnoreCase("LinearTrigger")) {
                                     t = new
LinearTrigger(args.get(0), args.get(1),
                                               args.get(2),
args.get(3));
                                     curArg = 4;
                                } else if
(type.equalsIgnoreCase("PlayerTrigger")) {
                                     t = new
PlayerTrigger(args.get(0), args.get(1),
                                               args.get(2),
args.get(3));
                                     curArg = 4;
                                }

     t.setTriggerDuration(args.get(curArg));
                                t.setDelay(args.get(curArg + 1));
                                int triggerable = 0;
                                if (args.size() <= curArg + 2) {
                                     if
(argsString.contains("checkpoint"))
                                          triggerable =
TriggerConstants.CHECKPOINT;
                                     else if
(argsString.contains("ender"))
                                          triggerable =
TriggerConstants.ENDER;
                                } else
                                     triggerable = args.get(curArg +
2);
                                t.setTriggerable(new
SavedTriggerable(triggerable));
                                elements.add(t);
                           }
                      }
                 }
```

```
			br.close();
			fr.close();
			connectSavedTriggers();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	private List<Integer> readArgs(String args) {
		List<Integer> result = new ArrayList<Integer>();
		args += " ";
		String current = "";
		for (int i = 0; i < args.length(); i++) {
			char ch = args.charAt(i);
			if ((ch >= '0' && ch <= '9') || ch == '-') {
				current += ch;
			} else {
				if (!current.isEmpty()) {
					result.add(Integer.parseInt(current));
					current = "";
				}
			}
		}
		return result;
	}

	private int readFallHeight(String line) {
		String fall = "";
		for (int i = 0; i < line.length(); i++) {
			char ch = line.charAt(i);
			if ((ch >= '0' && ch <= '9') || ch == '-') {
				fall += ch;
			}
		}
		return Integer.parseInt(fall);
	}

	private Point readSpawnPoint(String line) {
		String spawn[] = new String[2];
		spawn[0] = "";
		spawn[1] = "";
		int current = 0;
		for (int i = 0; i < line.length(); i++) {
			char ch = line.charAt(i);
			if ((ch >= '0' && ch <= '9') || ch == '-') {
				spawn[current] += ch;
			} else {
				current++;
				if (current > 1)
					break;
			}
		}
		return new Point(Integer.parseInt(spawn[0]),
Integer.parseInt(spawn[1]));
	}

	private int readTool(String line) {
		if (line.equalsIgnoreCase("boomerang"))
```

```
                  return ToolConstants.BOOMERANG;
            else if (line.equalsIgnoreCase("rope"))
                  return ToolConstants.ROPE;
            return ToolConstants.NONE;
      }
}
package com.dungeonescape.game;

import java.util.List;

import com.dungeonescape.common.ContactConstants;
import com.dungeonescape.element.GameElement;

public class PhysicsEngine {
      public void timestep(double d, List<GameElement> elements, double
gravity,
                  double friction) {
            // a setting to improve game pacing
            d = d / 2;

            for (int i = 0; i < elements.size(); i++) {
                  GameElement e1 = elements.get(i);
                  // object specific actions to be taken
                  e1.timestep(d, elements);

                  if (!e1.isFixed()) {
                        // apply friction and gravity depending on object
type
                        if (e1.isFricted())
                              applyFriction(e1, d, friction);
                        if (!e1.isFlying() && e1.getVerticalSpeed() < 20)
                              e1.setVerticalSpeed(e1.getVerticalSpeed() +
gravity * d);

                        // move object depending on its existing velocity
                        e1.moveX(e1.getHorizontalSpeed() * d);
                        e1.moveY(e1.getVerticalSpeed() * d);

                        // check the collision with each object and
handle the possible
                        // collisions
                        for (int n = 0; n < elements.size(); n++) {
                              GameElement e2 = elements.get(n);
                              if (!(e2 == e1)) {
                                    detectAndHandleCollision(e1, e2, d);
                              }
                        }
                  }

                  // remove inactive elements
                  if (!e1.isActive()) {
                        elements.remove(e1);
                        i--;
                  }
            }
      }
```

```
	public void detectAndHandleCollision(GameElement e1, GameElement
e2,
			double d) {
		if (e1.intersects(e2)) {
			e1.moveY(-e1.getVerticalSpeed() * d);
			if (!e1.intersects(e2)) {
				if (e1.getVerticalSpeed() > 0) {
					e1.contact(ContactConstants.BOTTOM, e2);
					e2.contact(ContactConstants.TOP, e1);
				} else {
					e1.contact(ContactConstants.TOP, e2);
					e2.contact(ContactConstants.BOTTOM, e1);
				}
				applyMomentum(e1, e2, true);
			} else {
				e1.moveY(e1.getVerticalSpeed() * d);
			}
		}
		if (e1.intersects(e2)) {
			e1.moveX(-e1.getHorizontalSpeed() * d);
			if (!e1.intersects(e2)) {
				if (e1.getHorizontalSpeed() > 0) {
					e1.contact(ContactConstants.RIGHT, e2);
					e2.contact(ContactConstants.LEFT, e1);
				} else {
					e1.contact(ContactConstants.LEFT, e2);
					e2.contact(ContactConstants.RIGHT, e1);
				}
				applyMomentum(e1, e2, false);
			} else {
				e1.moveX(e1.getHorizontalSpeed() * d);
			}
		}
		if (e1.intersects(e2)) {
			e1.contact(ContactConstants.IN, e2);
			e2.contact(ContactConstants.IN, e1);
		}
	}

	public void applyFriction(GameElement e, double d, double
friction) {
		if (e.getHorizontalSpeed() > friction)
			e.setHorizontalSpeed(e.getHorizontalSpeed() - friction
* d);
		else if (e.getHorizontalSpeed() < -1 * friction)
			e.setHorizontalSpeed(e.getHorizontalSpeed() + friction
* d);
		else
			e.setHorizontalSpeed(0);
		if (e.getVerticalSpeed() > friction)
			e.setVerticalSpeed(e.getVerticalSpeed() - friction *
d);
		else if (e.getVerticalSpeed() < -1 * friction)
			e.setVerticalSpeed(e.getVerticalSpeed() + friction *
d);
		else
			e.setVerticalSpeed(0);
	}
```

```
      public void applyMomentum(GameElement e1, GameElement e2, boolean
isVertical) {
            if (isVertical) {
                  if (e2.isFixed()) {
                        e1.setVerticalSpeed(-e1.getVerticalSpeed() *
e1.getElasticity());
                  } else {
                        double ratio = e1.getWeight() / e2.getWeight();
                        double tmp = e1.getVerticalSpeed();
                        e1.setVerticalSpeed((e2.getVerticalSpeed() - e1
                                    .getVerticalSpeed()) * ratio *
e1.getElasticity());
                        e2.setVerticalSpeed((tmp - e2.getVerticalSpeed())
* (1 / ratio)
                                    * e2.getElasticity());
                  }
            } else {
                  if (e2.isFixed()) {
                        e1.setHorizontalSpeed(-e1.getHorizontalSpeed()
                                    * e1.getElasticity());
                  } else {
                        double ratio = e1.getWeight() / e2.getWeight();
                        double tmp = e1.getHorizontalSpeed();
                        e1.setHorizontalSpeed((e2.getHorizontalSpeed() -
e1
                                    .getHorizontalSpeed()) * ratio *
e1.getElasticity());
                        e2.setHorizontalSpeed((tmp -
e2.getHorizontalSpeed())
                                    * (1 / ratio) * e2.getElasticity());
                  }
            }
      }
}
```

```
package com.dungeonescape.gameio;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Composite;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.font.FontRenderContext;
import java.awt.font.GlyphVector;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Properties;

import javax.imageio.ImageIO;
import javax.swing.JPanel;

import com.dungeonescape.game.Game;
import com.dungeonescape.ui.GameMenu;

public class GamePanel extends JPanel {
      private static final long serialVersionUID = -
7233967302635631295L;

      public static int rightButton = KeyEvent.VK_D;
      public static int leftButton = KeyEvent.VK_A;
      public static int jumpButton = KeyEvent.VK_W;
      public static int useButton = KeyEvent.VK_E;
      public static int restartButton = KeyEvent.VK_R;

      public static void loadKeyBindings() {
            Properties props = new Properties();
            InputStream is = null;

            File f = new File("keys.properties");
            if (f.exists()) {
                  try {
                        is = new FileInputStream(f);
                  } catch (FileNotFoundException e) {
                        e.printStackTrace();
                  }
                  try {
                        props.load(is);
                  } catch (IOException e) {
                        e.printStackTrace();
                  }
                  rightButton =
Integer.parseInt(props.getProperty("rightButton")
                              .toString());
                  leftButton =
Integer.parseInt(props.getProperty("leftButton")
                              .toString());
                  jumpButton =
Integer.parseInt(props.getProperty("jumpButton")
                              .toString());
                  useButton =
Integer.parseInt(props.getProperty("useButton")
                              .toString());
                  restartButton =
Integer.parseInt(props.getProperty("restartButton")
                              .toString());
            } else
                  saveKeyBindings();
      }

      public static void saveKeyBindings() {
            Properties props = new Properties();
            props.setProperty("rightButton", "" + rightButton);
            props.setProperty("leftButton", "" + leftButton);
            props.setProperty("jumpButton", "" + jumpButton);
            props.setProperty("useButton", "" + useButton);
            props.setProperty("restartButton", "" + restartButton);
            File f = new File("keys.properties");
            OutputStream out = null;
            try {
                  out = new FileOutputStream(f);
            } catch (FileNotFoundException e) {
                  e.printStackTrace();
            }
            try {
                  props.store(out, "This is an optional header comment
string");
            } catch (IOException e) {
                  e.printStackTrace();
            }
      }

      private Game game;
      private Point mousePosition;
      private Point cameraPosition;
      private BufferedImage cursorImage;
      private String tip;
      private GameMenu gameMenu;

      public GamePanel() {
            GameMouse mouse = new GameMouse();
            setFocusable(true);
            addKeyListener(new GameKey());
            addMouseListener(mouse);
            addMouseMotionListener(mouse);
            mousePosition = new Point(this.getSize().width / 2,
                        this.getSize().height / 2);
            cameraPosition = new Point(0, 0);
            try {
                  setCursorImage(ImageIO.read(new
File("img/cursor.png")));
            } catch (IOException e) {
                  e.printStackTrace();
            }
            setCursor(getToolkit().createCustomCursor(
                        new BufferedImage(3, 3,
BufferedImage.TYPE_INT_ARGB),
                        new Point(0, 0), "null"));
            setBackground(Color.white);
      }

      public void end(boolean success) {
            (new EnderThread(success)).start();
      }

      private class EnderThread extends Thread {
            private boolean success;

            public EnderThread(boolean success) {
                  this.success = success;
            }

            public void run() {
                  game.stop();
                  if (!success) {
                        game.reload();
                        dismissTip();
                        game.start();
                  } else if (gameMenu != null) {
                        gameMenu.levelComplete(GamePanel.this);
                  }
            }
      }

      public Game getGame() {
            return game;
      }

      public void setGame(Game game) {
            this.game = game;
      }

      @Override
      public void paint(Graphics g) {
            super.paint(g);

            refreshCameraPosition();
            game.getLevel().paintBackground((Graphics2D) g,
cameraPosition,
                        getSize());
            game.paint(g, cameraPosition);
            g.drawImage(cursorImage, mousePosition.x, mousePosition.y,
null);
            if (tip != null) {
                  Dimension size = getSize();
```

```
				Graphics2D g2d = ((Graphics2D) g);
				Composite prev = g2d.getComposite();
				g2d.setComposite(AlphaComposite.getInstance(
						AlphaComposite.SRC_OVER, 0.9f));
				g.fillRect(0, 0, size.width, size.height);
				g2d.setComposite(prev);
				g.setColor(Color.white);
				g.setFont(new Font("Helvetica", Font.BOLD, 32));
				int centerX = getWidth() / 2;
				int centerY = getHeight() / 2;
				FontMetrics fontMetrics = g.getFontMetrics();
				Rectangle stringBounds =
fontMetrics.getStringBounds(tip, g)
						.getBounds();
				Font font = g.getFont();
				FontRenderContext renderContext = ((Graphics2D) g)
						.getFontRenderContext();
				GlyphVector glyphVector = font
						.createGlyphVector(renderContext, tip);
				Rectangle visualBounds =
glyphVector.getVisualBounds().getBounds();
				int textX = centerX - stringBounds.width / 2;
				int textY = centerY - visualBounds.height / 2 -
visualBounds.y;

				g.drawString(tip, textX, textY);
			}
		}

		public void refreshCameraPosition() {
			int width = getSize().width;
			int height = getSize().height;
			cameraPosition.x = ((mousePosition.x + cameraPosition.x + 2
* (game
					.getPlayerPosition().x)) / 3) - (width / 2);
			cameraPosition.y = ((mousePosition.y + cameraPosition.y + 2
* (game
					.getPlayerPosition().y)) / 3) - (height / 2);
		}

		public BufferedImage getCursorImage() {
			return cursorImage;
		}

		public void setCursorImage(BufferedImage cursorImage) {
			this.cursorImage = cursorImage;
		}

		public void showTip(String tip) {
			if (!tip.isEmpty()) {
				this.tip = tip;
			}
		}

		public void dismissTip() {
			tip = null;
		}
```

```
	public GameMenu getGameMenu() {
		return gameMenu;
	}

	public void setGameMenu(GameMenu gameMenu) {
		this.gameMenu = gameMenu;
	}

	public class GameKey extends KeyAdapter {
		public void keyPressed(KeyEvent e) {
			dismissTip();
			if (e.getKeyCode() == rightButton) {
				game.right(true);
			}
			if (e.getKeyCode() == leftButton) {
				game.left(true);
			}
			if (e.getKeyCode() == jumpButton) {
				game.jump(true);
			}
			if (e.getKeyCode() == useButton) {
				game.use();
			}
			if (e.getKeyCode() == KeyEvent.VK_ESCAPE) {
				if (game.isStopped())
					game.start();
				else
					game.stop();
			}
			if (e.getKeyCode() == restartButton) {
				end(false);
			}
		}

		public void keyReleased(KeyEvent e) {
			dismissTip();
			if (e.getKeyCode() == rightButton) {
				game.right(false);
			}
			if (e.getKeyCode() == leftButton) {
				game.left(false);
			}
			if (e.getKeyCode() == jumpButton) {
				game.jump(false);
			}
		}
	}

	public class GameMouse extends MouseAdapter {
		public void mousePressed(MouseEvent e) {
			dismissTip();
			if (e.getButton() == MouseEvent.BUTTON1)
				game.useTool(e.getX() + cameraPosition.x,
e.getY()
						+ cameraPosition.y);
		}

		public void mouseMoved(MouseEvent e) {
```

```
                mousePosition.x = (e.getX());
                mousePosition.y = (e.getY());
                game.getPlayer().setDirection(mousePosition.x >
getWidth() / 2);

            }

            public void mouseDragged(MouseEvent e) {
                mouseMoved(e);
            }
        }
}
package com.dungeonescape.gameio;

import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;

import javax.swing.AbstractAction;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JRootPane;
import javax.swing.KeyStroke;

public class ToggleFullscreen extends JFrame {
        private static final long serialVersionUID = 6075801570706118769L;

        private boolean fullscreen;

        // private Point prevLocation;
        // private Dimension prevSize;

        public ToggleFullscreen() {
            super();
            setSize(800, 600);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            fullscreen = false;
            // prevLocation = getLocation();
            JRootPane rootPane = getRootPane();
            rootPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
                    KeyStroke.getKeyStroke(KeyEvent.VK_F11, 0),
"goFullscreen");
            rootPane.getActionMap().put("goFullscreen", new
AbstractAction() {
                private static final long serialVersionUID =
9145906117511743707L;

                public void actionPerformed(ActionEvent arg0) {
                    toggle();
                }
            });
            setVisible(true);
        }

        public void toggle() {
            // dispose();
            // if (!fullscreen) {
            // prevLocation = getLocation();
            // prevSize = getSize();
```

```
			// setLocation(0, 0);
			// setUndecorated(true);
			// setSize(Toolkit.getDefaultToolkit().getScreenSize());
			// } else {
			// setLocation(prevLocation);
			// setUndecorated(false);
			// setSize(prevSize);
			// }
			// setVisible(true);
			if (!fullscreen)
				setExtendedState(JFrame.MAXIMIZED_BOTH);
			else
				setExtendedState(JFrame.NORMAL);
			fullscreen = !fullscreen;
		}

		public boolean isFullscreen() {
			return fullscreen;
		}
}
package com.dungeonescape.gameio.editor;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JPanel;

import com.dungeonescape.element.Button;
import com.dungeonescape.element.ContactTrigger;
import com.dungeonescape.element.Door;
import com.dungeonescape.element.Lever;
import com.dungeonescape.element.LinearTrigger;
import com.dungeonescape.element.MoveableObject;
import com.dungeonescape.element.Obstacle;
import com.dungeonescape.element.Platform;
import com.dungeonescape.element.PlayerTrigger;
import com.dungeonescape.element.Trigger;
import com.dungeonescape.ui.GameButton;

public class AddElementPanel extends JPanel implements ActionListener {
		private static final long serialVersionUID = -
2835564701992806419L;

		private EditorSidebar es;

		public AddElementPanel(EditorSidebar es) {
			this.es = es;
			setLayout(new GridLayout(0, 1));
			add(generateButton("Obstacle"));
			add(generateButton("MoveableObject"));
			add(generateButton("Door"));
			add(generateButton("Platform"));
			add(generateButton("ContactTrigger"));
			add(generateButton("Button"));
			add(generateButton("Lever"));
			add(generateButton("PlayerTrigger"));
			add(new SaveBar(es.getEditorPanel()));
```

```
	}

	private GameButton generateButton(String s) {
		GameButton result = new GameButton(s);
		result.addActionListener(this);
		return result;
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		String type = ((GameButton) e.getSource()).getText();
		if (type.equals("Obstacle")) {
			es.addElement(new Obstacle(0, 0, 100, 100));
		} else if (type.equals("MoveableObject")) {
			es.addElement(new MoveableObject(0, 0));
		} else if (type.equals("Door")) {
			es.addElement(new Door(0, 0));
		} else if (type.equals("Platform")) {
			es.addElement(new Platform(0, 0, 100, 100, 100,
false));
		} else if (type.equals("Trigger")) {
			es.addElement(new Trigger(0, 0));
		} else if (type.equals("ContactTrigger")) {
			es.addElement(new ContactTrigger(0, 0, 100, 100));
		} else if (type.equals("Button")) {
			es.addElement(new Button(0, 0));
		} else if (type.equals("Lever")) {
			es.addElement(new Lever(0, 0));
		} else if (type.equals("LinearTrigger")) {
			es.addElement(new LinearTrigger(0, 0, 100, 100));
		} else if (type.equals("PlayerTrigger")) {
			es.addElement(new PlayerTrigger(0, 0, 100, 100));
		}
	}
}
package com.dungeonescape.gameio.editor;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.geom.Rectangle2D;
import java.util.List;

import javax.swing.JPanel;

import com.dungeonescape.element.GameElement;

public class EditorMainPanel extends JPanel {
	private static final long serialVersionUID = -
4432837109006093617L;

	private List<GameElement> elements;
	private Point mouse, camera, prevMouse;
	private EditorPanel ep;
	private boolean clicked;

	public EditorMainPanel(List<GameElement> elements, EditorPanel ep)
{
		setPreferredSize(new Dimension(1200, 800));
		this.ep = ep;
		this.elements = elements;
		camera = new Point(0, 0);
		addMouseListener(new EditorMouse());
		addMouseMotionListener(new EditorMouse());
		clicked = false;
		mouse = new Point(0, 0);
		setBackground(Color.white);
	}

	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		ep.getLevel().paintBackground((Graphics2D) g, camera,
getSize());
		Point spawn = ep.getLevel().getSpawnPoint();
		for (int i = 0; i < elements.size(); i++) {
			GameElement e = elements.get(i);
			e.draw(g, camera);
		}
		g.drawString(mouse.x + ", " + mouse.y, 50, 50);
		Color prevColor = g.getColor();
		g.setColor(Color.yellow);
		g.drawLine(-10 - camera.x, 0 - camera.y, 10 - camera.x, 0 -
camera.y);
		g.drawLine(0 - camera.x, 10 - camera.y, 0 - camera.x, -10 -
camera.y);
		g.setColor(Color.orange);
		g.drawLine(-10 - camera.x + spawn.x, 0 - camera.y + spawn.y,
10
				- camera.x + spawn.x, 0 - camera.y + spawn.y);
		g.drawLine(0 - camera.x + spawn.x, 10 - camera.y + spawn.y,
0
				- camera.x + spawn.x, -10 - camera.y + spawn.y);
		g.drawString("Spawn Point", spawn.x - camera.x, spawn.y -
camera.y);
		g.setColor(Color.cyan);
		g.drawLine(0, ep.getLevel().getFallHeight() - camera.y,
getWidth(), ep.getLevel()
				.getFallHeight() - camera.y);
		g.drawString("Fall Height", 0, ep.getLevel().getFallHeight()
- camera.y);
		g.setColor(prevColor);
	}

	public Point getMouse() {
		return mouse;
	}

	private class EditorMouse implements MouseListener,
MouseMotionListener {
		@Override
		public void mouseDragged(MouseEvent e) {
```

```
                        mouseMoved(e);
                  }

                  @Override
                  public void mouseMoved(MouseEvent e) {
                        mouse.x = camera.x + e.getX();
                        mouse.y = camera.y + e.getY();
                        GameElement element = ep.getSelectedElement();
                        if (clicked) {
                              if (element != null) {
                                    int xdiff = e.getX() - prevMouse.x;
                                    int ydiff = e.getY() - prevMouse.y;
                                    prevMouse = e.getPoint();
                                    element.moveX(xdiff);
                                    element.moveY(ydiff);
                              } else {
                                    int xdiff = e.getX() - prevMouse.x;
                                    int ydiff = e.getY() - prevMouse.y;
                                    prevMouse = e.getPoint();
                                    camera.x -= (xdiff);
                                    camera.y -= (ydiff);
                              }
                        }
                        ep.refresh();
                  }

                  @Override
                  public void mouseClicked(MouseEvent e) {

                  }

                  @Override
                  public void mouseEntered(MouseEvent e) {

                  }

                  @Override
                  public void mouseExited(MouseEvent e) {

                  }

                  @Override
                  public void mousePressed(MouseEvent e) {
                        clicked = true;
                        prevMouse = e.getPoint();
                        List<GameElement> elements =
ep.getLevel().getElements();
                        for (int i = 0; i < elements.size(); i++) {
                              GameElement element = elements.get(i);
                              Rectangle2D rect = element.getRectangle();
                              boolean has = false;
                              if (rect != null)
                                    has = rect.contains(mouse);
                              else
                                    has = element.getLine().contains(mouse);
                              if (has) {
                                    ep.setSelectedElement(element, i);
                                    return;
```

```
				}
			}
			ep.setSelectedElement(null, -10);
		}

		@Override
		public void mouseReleased(MouseEvent e) {
			clicked = false;
			ep.setSelectedElement(ep.getSelectedElement(),
					ep.getSelectedElementId());
		}
	}
}
package com.dungeonescape.gameio.editor;

import java.awt.BorderLayout;

import javax.swing.JPanel;

import com.dungeonescape.element.GameElement;
import com.dungeonescape.game.Level;
import com.dungeonescape.gameio.ToggleFullscreen;

public class EditorPanel extends JPanel {
	private static final long serialVersionUID = 864102639139728889L;

	private EditorSidebar es;
	private EditorMainPanel emp;
	private Level level;

	public EditorPanel() {
		setLevel(new Level());
		setLayout(new BorderLayout());
		es = new EditorSidebar(this);
		emp = new EditorMainPanel(level.getElements(), this);
		add(emp, BorderLayout.CENTER);
		add(es, "East");
		es.setSelectedGameElement(null, -10);
	}

	public void addElement(GameElement e) {
		level.addElement(e);
		refresh();
	}

	public void refresh() {
		emp.updateUI();
	}

	public Level getLevel() {
		return level;
	}

	public void setLevel(Level level) {
		this.level = level;
	}

	public void setSelectedElement(GameElement e, int id) {
		es.setSelectedGameElement(e, id);
	}

	public GameElement getSelectedElement() {
		return es.getSelectedGameElement();
	}

	public int getSelectedElementId() {
		return es.getSelectedGameElementId();
	}

	public static void main(String[] args) {
		EditorPanel ep = new EditorPanel();
		ToggleFullscreen f = new ToggleFullscreen();
		f.add(ep);
		f.setVisible(true);
	}
}
package com.dungeonescape.gameio.editor;

import java.awt.BorderLayout;

import javax.swing.JPanel;

import com.dungeonescape.element.GameElement;

public class EditorSidebar extends JPanel {
	private static final long serialVersionUID = 4636906374046518685L;

	private EditorPanel ep;
	private ElementPropertiesPanel epp;
	private AddElementPanel aep;

	public EditorSidebar(EditorPanel ep) {
		this.setEditorPanel(ep);
		setLayout(new BorderLayout());
		epp = new ElementPropertiesPanel(ep);
		aep = new AddElementPanel(this);
		add(aep, "South");
		add(epp, "North");
	}

	public void setSelectedGameElement(GameElement
selectedGameElement, int id) {
		epp.setSelectedGameElement(selectedGameElement, id);
	}

	public GameElement getSelectedGameElement() {
		return epp.getSelectedGameElement();
	}

	public void addElement(GameElement e) {
		ep.addElement(e);
	}

	public EditorPanel getEditorPanel() {
		return ep;
	}
	public void setEditorPanel(EditorPanel ep) {
		this.ep = ep;
	}

	public int getSelectedGameElementId() {
		return epp.getSelectedElementId();
	}
}
package com.dungeonescape.gameio.editor;

import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.LinkedList;
import java.util.List;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

import com.dungeonescape.common.ToolConstants;
import com.dungeonescape.element.Button;
import com.dungeonescape.element.ContactTrigger;
import com.dungeonescape.element.Door;
import com.dungeonescape.element.GameElement;
import com.dungeonescape.element.Lever;
import com.dungeonescape.element.LinearTrigger;
import com.dungeonescape.element.MoveableObject;
import com.dungeonescape.element.Obstacle;
import com.dungeonescape.element.Platform;
import com.dungeonescape.element.PlayerTrigger;
import com.dungeonescape.element.SavedTriggerable;
import com.dungeonescape.element.Trigger;
import com.dungeonescape.element.Triggerable;
import com.dungeonescape.ui.GameButton;

public class ElementPropertiesPanel extends JPanel {
	private static final long serialVersionUID = -
5207694422232205757L;

	private GameElement element;
	private int elementId;
	private EditorPanel ep;

	public ElementPropertiesPanel(EditorPanel ep) {
		this.ep = ep;
		setLayout(new GridLayout(0, 1));
		elementId = -10;
	}

	public void setSelectedGameElement(GameElement
selectedGameElement, int id) {
		this.element = selectedGameElement;
		elementId = id;
		refresh();
	}
	public void refresh() {
		removeAll();
		List<JPanel> fields = generateFields();
		for (JPanel tf : fields)
			add(tf);
		updateUI();
	}

	public GameElement getSelectedGameElement() {
		return element;
	}

	private JPanel generateField(String label, String defaultValue,
			ActionListener al) {
		JTextField tf = new JTextField(defaultValue);
		tf.addActionListener(al);
		tf.setPreferredSize(new Dimension(50, 25));
		JPanel result = new JPanel();
		result.add(new JLabel(label + ": "));
		result.add(tf);
		return result;
	}

	private List<JPanel> generateFields() {
		List<JPanel> result = new LinkedList<JPanel>();

		if (element != null) {
			GameButton removeButton = new GameButton("Remove
Element");
			removeButton.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent event) {
					ep.getLevel().removeElement(element);
					if (element instanceof Triggerable) {
						for (GameElement e :
ep.getLevel().getElements()) {
							if (e instanceof Trigger) {
								Trigger t = (Trigger) e;
								if (t.getTriggerable() ==
element) {

	t.setTriggerable(null);
								}
							}
						}
					}
					element = null;
					ep.refresh();
				}
			});
			JPanel removePanel = new JPanel();
			removePanel.add(removeButton);
			result.add(removePanel);
			result.add(generateField("x", "" + (int)
element.getX(),
					new ActionListener() {
						public void
actionPerformed(ActionEvent e) {
      element.setX(Integer.parseInt(((JTextField) (e

      .getSource())).getText()));
                                    ep.refresh();
                              }
                        }));
                  result.add(generateField("y", "" + (int)
element.getY(),
                        new ActionListener() {
                              public void
actionPerformed(ActionEvent e) {

      element.setY(Integer.parseInt(((JTextField) (e

      .getSource())).getText()));
                                    ep.refresh();
                              }
                        }));

                  if (element instanceof Obstacle) {
                        result.add(generateField("width", "" +
element.getWidth(),
                              new ActionListener() {
                                    public void
actionPerformed(ActionEvent e) {
                                          element.setWidth(Integer

      .parseInt(((JTextField) (e.getSource()))

      .getText()));
                                          ep.refresh();
                                    }
                              }));
                        result.add(generateField("height", "" +
element.getHeight(),
                              new ActionListener() {
                                    public void
actionPerformed(ActionEvent e) {
                                          element.setHeight(Integer

      .parseInt(((JTextField) (e.getSource()))

      .getText()));
                                          ep.refresh();
                                    }
                              }));
                  } else if (element instanceof MoveableObject) {
                  } else if (element instanceof Door) {
                  } else if (element instanceof Platform) {
                        result.add(generateField("width", "" +
element.getWidth(),
                              new ActionListener() {
                                    public void
actionPerformed(ActionEvent e) {
                                          element.setWidth(Integer

      .parseInt(((JTextField) (e.getSource()))
      .getText()));
                                          ep.refresh();
                                    }
                              }));
                        result.add(generateField("height", "" +
element.getHeight(),
                              new ActionListener() {
                                    public void
actionPerformed(ActionEvent e) {
                                          element.setHeight(Integer

      .parseInt(((JTextField) (e.getSource()))

      .getText()));
                                          ep.refresh();
                                    }
                              }));
                        result.add(generateField("maxTravel",
                              "" + ((Platform)
element).getMaxTravel(),
                              new ActionListener() {
                                    public void
actionPerformed(ActionEvent e) {
                                          ((Platform)
element).setMaxTravel(Integer

      .parseInt(((JTextField) (e.getSource()))

      .getText()));
                                          ep.refresh();
                                    }
                              }));
                        result.add(generateField("verticalTravel", ""
                              + ((Platform)
element).isVerticalTravel(),
                              new ActionListener() {
                                    public void
actionPerformed(ActionEvent e) {
                                          ((Platform)
element).setVerticalTravel(Boolean

      .parseBoolean(((JTextField) (e

      .getSource())).getText()));
                                          ep.refresh();
                                    }
                              }));
                  } else if (element instanceof Trigger) {
                        if (element instanceof ContactTrigger) {
                              result.add(generateField("width", "" +
element.getWidth(),
                                    new ActionListener() {
                                          public void
actionPerformed(ActionEvent e) {

      element.setWidth(Integer
      .parseInt(((JTextField) (e

      .getSource())).getText()));
                                                ep.refresh();
                                          }
                                    }));
                              result.add(generateField("height",
                                    "" + element.getHeight(), new
ActionListener() {
                                          public void
actionPerformed(ActionEvent e) {

      element.setHeight(Integer

      .parseInt(((JTextField) (e

      .getSource())).getText()));
                                                ep.refresh();
                                          }
                                    }));
                        } else if (element instanceof Button) {
                        } else if (element instanceof Lever) {
                        } else if (element instanceof LinearTrigger) {
                        } else if (element instanceof PlayerTrigger) {
                              result.add(generateField("width", "" +
element.getWidth(),
                                    new ActionListener() {
                                          public void
actionPerformed(ActionEvent e) {

      element.setWidth(Integer

      .parseInt(((JTextField) (e

      .getSource())).getText()));
                                                ep.refresh();
                                          }
                                    }));
                              result.add(generateField("height",
                                    "" + element.getHeight(), new
ActionListener() {
                                          public void
actionPerformed(ActionEvent e) {

      element.setHeight(Integer

      .parseInt(((JTextField) (e

      .getSource())).getText()));
                                                ep.refresh();
                                          }
                                    }));
                        } else {
                        }
                        result.add(generateField("triggerDuration", ""
                                    + ((Trigger)
element).getTriggerDuration(),
                                    new ActionListener() {
                                          public void
actionPerformed(ActionEvent e) {
                                                ((Trigger)
element).setTriggerDuration(Integer

      .parseInt(((JTextField) (e.getSource()))

      .getText()));
                                                ep.refresh();
                                          }
                                    }));
                        result.add(generateField("delay",
                                    "" + ((Trigger) element).getDelay(),
                                    new ActionListener() {
                                          public void
actionPerformed(ActionEvent e) {
                                                ((Trigger)
element).setDelay(Integer

      .parseInt(((JTextField) (e.getSource()))

      .getText()));
                                                ep.refresh();
                                          }
                                    }));
                        result.add(generateField("triggerable", "",
                                    new ActionListener() {
                                          public void
actionPerformed(ActionEvent e) {
                                                ((Trigger)
element).setTriggerable(new SavedTriggerable(

      Integer.parseInt(((JTextField) (e

      .getSource())).getText())));

      ep.getLevel().connectSavedTriggers();
                                                ep.refresh();
                                          }
                                    }));
                  }
                  if (element instanceof Triggerable) {
                        JPanel panel = new JPanel();
                        panel.add(new JLabel("Triggerable No: " +
elementId));
                        result.add(panel);
                  }
            } else {
                  result.add(generateField("fallHeight", ""
                              + ep.getLevel().getFallHeight(), new
ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                              ep.getLevel().setFallHeight(
                                          Integer.parseInt(((JTextField)
(e.getSource()))
                                                      .getText()));
                        }
```

```
                 }));
                 int currentTool = ep.getLevel().getTool();
                 String currentToolString;
                 if (currentTool == ToolConstants.BOOMERANG)
                       currentToolString = "boomerang";
                 else if (currentTool == ToolConstants.ROPE)
                       currentToolString = "rope";
                 else
                       currentToolString = "none";
                 result.add(generateField("tool", currentToolString,
                            new ActionListener() {
                                  public void
actionPerformed(ActionEvent e) {
                                        String tool = ((JTextField)
(e.getSource()))
                                                   .getText();
                                        int toolInt;
                                        if
(tool.equalsIgnoreCase("boomerang"))
                                             toolInt =
ToolConstants.BOOMERANG;
                                        else if
(tool.equalsIgnoreCase("rope"))
                                             toolInt =
ToolConstants.ROPE;
                                        else
                                             toolInt =
ToolConstants.NONE;
                                        ep.getLevel().setTool(toolInt);
                                  }
                            }));
                 result.add(generateField("Spawn X", ""
                            + ep.getLevel().getSpawnPoint().x, new
ActionListener() {
                       public void actionPerformed(ActionEvent e) {
                            ep.getLevel().getSpawnPoint().x = (Integer
                                       .parseInt(((JTextField)
(e.getSource())).getText()));
                       }
                 }));
                 result.add(generateField("Spawn Y", ""
                            + ep.getLevel().getSpawnPoint().y, new
ActionListener() {
                       public void actionPerformed(ActionEvent e) {
                            ep.getLevel().getSpawnPoint().y = (Integer
                                       .parseInt(((JTextField)
(e.getSource())).getText()));
                       }
                 }));
                 result.add(generateField("tip", ep.getLevel().getTip(),
                            new ActionListener() {
                                  public void
actionPerformed(ActionEvent e) {
                                        ep.getLevel().setTip(
                                                   ((JTextField)
(e.getSource())).getText());
                                  }
                            }));
```

```
                result.add(generateField("backgroundUrl", ep.getLevel()
                        .getBackgroundUrl(), new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        ep.getLevel().setBackgroundUrl(
                                ((JTextField)
(e.getSource())).getText());
                    }
                }));
            }
            return result;
        }

        public int getSelectedElementId() {
            return elementId;
        }
}
package com.dungeonescape.gameio.editor;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JPanel;
import javax.swing.JTextField;

import com.dungeonescape.ui.GameButton;

public class SaveBar extends JPanel implements ActionListener {
        private static final long serialVersionUID = 7496162042016776991L;

        private EditorPanel ep;
        private JTextField tf;

        public SaveBar(EditorPanel ep) {
            this.ep = ep;
            tf = new JTextField();
            tf.setPreferredSize(new Dimension(100, 25));
            add(tf);
            GameButton save = new GameButton("Save");
            save.addActionListener(this);
            add(save);
            GameButton load = new GameButton("Load");
            load.addActionListener(this);
            add(load);
        }

        @Override
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == this.getComponent(1))
                ep.getLevel().saveLevel(new File("levels/" +
tf.getText() + ".level"));
            if (e.getSource() == this.getComponent(2))
                ep.getLevel().loadLevel(new File("levels/" +
tf.getText() + ".level"));
        }
}
```

```
package com.dungeonescape.tool;

import com.dungeonescape.element.Boomerang;
import com.dungeonescape.element.GameElement;

public class BoomerangTool extends Tool {
      private Boomerang boomerang;
      private double speed;

      public BoomerangTool() {
            boomerang = new Boomerang();
            speed = 35;
      }

      @Override
      public void timestep(double d) {
            if (boomerang.isActive()) {
                  double speedRatio = Math.abs(1
                              - Math.hypot(x - boomerang.getX(), y -
boomerang.getY())
                              / boomerang.getRange());
                  if (speedRatio < 0.3)
                        speedRatio = 0.3;
                  if (boomerang.isReturning()) {
                        double angle = Math.atan((double) (y -
boomerang.getY())
                                    / (double) (x - boomerang.getX()));
                        if (x - boomerang.getX() < 0)
                              angle = Math.PI + angle;
                        boomerang
                                    .setVerticalSpeed(Math.sin(angle) *
speed * speedRatio);
                        boomerang.setHorizontalSpeed(Math.cos(angle) *
speed
                                    * speedRatio);
                  } else {

      boomerang.setVerticalSpeed(Math.sin(boomerang.getAngle())
                                    * speed * speedRatio);

      boomerang.setHorizontalSpeed(Math.cos(boomerang.getAngle())
                                    * speed * speedRatio);
                  }
            }

      }

      @Override
      public GameElement use(int xDest, int yDest) {
            if (!boomerang.isActive()) {
                  double angle = Math.atan((double) (y - yDest)
                              / (double) (x - xDest));
                  if (x - xDest < 0)
                        boomerang = new Boomerang(x, y, angle, speed,
this);
                  else {
                        angle = Math.PI + angle;
                        boomerang = new Boomerang(x, y, angle, speed,
this);
                  }
                  return boomerang;
            }
            return null;
      }

      public Boomerang getBoomerang() {
            return boomerang;
      }

      public void setBoomerang(Boomerang boomerang) {
            this.boomerang = boomerang;
      }
}
package com.dungeonescape.tool;

import com.dungeonescape.element.GameElement;
import com.dungeonescape.element.Hook;

public class Rope extends Tool {
      private double reach;
      private double speed, pullSpeed;
      private Hook hook;

      public Rope() {
            speed = 20;
            reach = 300;
            hook = new Hook();
            pullSpeed = 20;
      }

      @Override
      public GameElement use(int xDest, int yDest) {
            if (!hook.isActive()) {
                  double angle = Math.atan((double) (y - yDest)
                              / (double) (x - xDest));
                  if (x - xDest < 0)
                        hook = new Hook(x, y, angle, speed, this);
                  else {
                        angle = Math.PI + angle;
                        hook = new Hook(x, y, angle, speed, this);
                  }
                  return hook;
            } else {
                  hook.setActive(false);
            }
            return null;
      }

      @Override
      public void timestep(double d) {
            if (hook.isActive() && hook.isGrappled()) {
                  double angle = Math.atan((double) (hook.getY() -
getY())
                              / (double) (hook.getX() - getX()));
                  if (hook.getX() - getX() <= 0)
```

```
                        angle = Math.PI + angle;
                  owner.setHorizontalSpeed(Math.cos(angle) *
getPullSpeed());
                  owner.setVerticalSpeed(Math.sin(angle) *
getPullSpeed());
            }
      }

      public double getReach() {
            return reach;
      }

      public void setReach(double reach) {
            this.reach = reach;
      }

      public double getSpeed() {
            return speed;
      }

      public void setSpeed(double speed) {
            this.speed = speed;
      }

      public Hook getHook() {
            return hook;
      }

      public void setHook(Hook hook) {
            this.hook = hook;
      }

      public double getPullSpeed() {
            return pullSpeed;
      }

      public void setPullSpeed(double pullSpeed) {
            this.pullSpeed = pullSpeed;
      }
}
package com.dungeonescape.tool;

import com.dungeonescape.element.GameElement;

public abstract class Tool {
      protected double x, y;
      protected GameElement owner;

      public Tool() {
      }

      public GameElement use(int xDest, int yDest) {
            return null;
      }

      public void timestep(double d) {
      }
```

```
	public double getX() {
		return x;
	}

	public void setX(double x) {
		this.x = x;
	}

	public double getY() {
		return y;
	}

	public void setY(double y) {
		this.y = y;
	}

	public GameElement getOwner() {
		return owner;
	}

	public void setOwner(GameElement owner) {
		this.owner = owner;
	}
}
```

```
package com.dungeonescape.ui;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;

import javax.swing.JButton;
import javax.swing.border.BevelBorder;

public class GameButton extends JButton {
      private static final long serialVersionUID = -
2935417191687118265L;

      private final Color PRESSED = new Color(128, 21, 21);
      private final Color DEFAULT = PRESSED.brighter();
      private final Color HOVERED = DEFAULT.brighter();

      public GameButton() {
            this("");
      }

      public GameButton(String text) {
            super(text);
            super.setContentAreaFilled(false);
            setBackground(DEFAULT);
            setForeground(Color.WHITE);
            setFocusPainted(false);
            setFont(new Font("Tahoma", Font.BOLD, 12));
            setBorder(new BevelBorder(BevelBorder.RAISED));
      }

      @Override
      public void paintComponent(Graphics g) {
            if (getModel().isPressed()) {
                  g.setColor(PRESSED);
            } else if (getModel().isRollover()) {
                  g.setColor(HOVERED);
            } else {
                  g.setColor(DEFAULT);
            }
            g.fillRect(0, 0, getWidth(), getHeight());
            super.paintComponent(g);
      }

      @Override
      public void setContentAreaFilled(boolean b) {
      }
}
package com.dungeonescape.ui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Insets;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FilenameFilter;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.AbstractAction;
import javax.swing.ImageIcon;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JRootPane;
import javax.swing.JTextArea;
import javax.swing.KeyStroke;
import javax.swing.ListSelectionModel;
import javax.swing.border.EmptyBorder;
import javax.swing.border.TitledBorder;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.plaf.FontUIResource;

import com.dungeonescape.common.Images;
import com.dungeonescape.game.Game;
import com.dungeonescape.gameio.GamePanel;
import com.dungeonescape.gameio.ToggleFullscreen;
import com.dungeonescape.gameio.editor.EditorPanel;

public class GameMenu extends JLayeredPane implements ComponentListener
{
      private static final long serialVersionUID = 4807364471121343782L;

      private final int AMOUNT_OF_PREMADE_LEVELS = 9;

      private GameDialog dialog;
      private BufferedImage background;

      public static void main(String[] args) {
            Images.loadImages();
            GamePanel.loadKeyBindings();
            System.setProperty("sun.java2d.opengl", "True");
            ToggleFullscreen f = new ToggleFullscreen();
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            GameMenu gm = new GameMenu(f.getRootPane());
            f.add(gm, BorderLayout.CENTER);
            f.setSize(1400, 1000);
      }
      public GameMenu(JRootPane rootPane) {
            dialog = null;
            try {
                  background = ImageIO.read(new
File("img/menu_back.png"));
            } catch (IOException e) {
                  e.printStackTrace();
            }
            setBackground(Color.black);
            rootPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
                        KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0),
"dialog");
            rootPane.getActionMap().put("dialog", new AbstractAction() {
                  private static final long serialVersionUID =
9145906117511743707L;

                  public void actionPerformed(ActionEvent e) {
                        if (getLayout() == null) {
                              if (getComponent(0) instanceof GamePanel) {
                                    if (!(dialog instanceof
LevelCompleteMenu))
                                          showHideGameDialog(new
PauseMenuWithRestart(
                                                      getSize(),
(GamePanel) (getComponent(0))));
                              } else
                                    showHideGameDialog(new
PauseMenu(getSize()));
                        }
                  }
            });
            addComponentListener(this);
            navigateTo(new MainMenu());
      }

      public void paintComponent(Graphics g) {
            g.drawImage(background, 0, 0, getWidth(), getHeight(),
this);
      }

      public void navigateTo(Component panel) {
            removeAll();
            dialog = null;
            if (!(panel instanceof GamePanel) && !(panel instanceof
EditorPanel)) {
                  setLayout(new GridBagLayout());
                  add(panel);
            } else {
                  setLayout(null);
                  panel.setSize(getSize());
                  panel.setLocation(new Point(0, 0));
                  add(panel, JLayeredPane.DEFAULT_LAYER);
            }
            revalidate();
            repaint();
      }

      public void showHideGameDialog(GameDialog panel) {
```

```
			if (dialog == null) {
				dialog = panel;
				add(panel, JLayeredPane.PALETTE_LAYER);
			} else {
				remove(dialog);
				dialog = null;
			}
			revalidate();
			repaint();
		}

		public void levelComplete(GamePanel src) {
			LevelCompleteMenu lcm = new LevelCompleteMenu(getSize(),
src);
			String level =
src.getGame().getLevel().getFile().getAbsolutePath();
			if (level.contains("/premade/")) {
				if
(Integer.parseInt(level.charAt(level.lastIndexOf('.') - 1) + "") <
AMOUNT_OF_PREMADE_LEVELS)
					lcm.setNextLevelEnabled(true);
			}
			showHideGameDialog(lcm);
		}

		public void componentResized(ComponentEvent e) {
			Component[] cmps = getComponents();
			for (int i = 0; i < cmps.length; i++) {
				if (cmps[i] != null) {
					if (cmps[i] instanceof GamePanel
							|| cmps[i] instanceof EditorPanel
							|| cmps[i] instanceof GameDialog) {
						cmps[i].setSize(getSize());
					}
				}
			}
		}

		private class MainMenu extends JPanel {
			private static final long serialVersionUID = -
8701274909161560797L;

			public MainMenu() {
				setBackground(new Color(0, 0, 0, 125));
				setLayout(new BorderLayout(10, 15));
				setBorder(new EmptyBorder(new Insets(25, 40, 40, 40)));
				Dimension buttonSize = new Dimension(200, 50);
				JLabel logo = new JLabel(new
ImageIcon("img/logo.png"));
				add(logo, "North");
				JPanel center = new JPanel();
				center.setLayout(new GridLayout(0, 1, 10, 10));
				center.setOpaque(false);
				GameButton play = new GameButton("Play");
				play.addActionListener(new ActionListener() {
					public void actionPerformed(ActionEvent e) {
						navigateTo(new PlayMenu());
					}
			});
			play.setPreferredSize(buttonSize);
			GameButton editor = new GameButton("Editor");
			editor.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					navigateTo(new EditorPanel());
				}
			});
			editor.setPreferredSize(buttonSize);
			GameButton options = new GameButton("Options");
			options.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					OptionsMenu om = new OptionsMenu();
					navigateTo(om);
					om.requestFocus();
				}
			});
			GameButton help = new GameButton("Help");
			help.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					navigateTo(new HelpPanel());
				}
			});
			help.setPreferredSize(buttonSize);
			options.setPreferredSize(buttonSize);
			GameButton credits = new GameButton("Credits");
			credits.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					navigateTo(new CreditsPanel());
				}
			});
			credits.setPreferredSize(buttonSize);
			GameButton exit = new GameButton("Exit");
			exit.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					System.exit(0);
				}
			});
			exit.setPreferredSize(buttonSize);
			add(center);
			center.add(play);
			center.add(editor);
			center.add(options);
			center.add(help);
			center.add(credits);
			center.add(exit);
		}
	}

	private class OptionsMenu extends JPanel implements
ActionListener,
			KeyListener {
		private static final long serialVersionUID = -
2411265021454995104L;

		private GameButton jump, left, right, use, restart,
selected;
```

```
            public OptionsMenu() {
                  setFocusable(true);
                  addKeyListener(this);
                  setBackground(new Color(0, 0, 0, 125));
                  setLayout(new GridLayout(0, 1, 10, 10));
                  setBorder(new EmptyBorder(new Insets(40, 40, 40, 40)));
                  Dimension buttonSize = new Dimension(200, 50);
                  jump = new GameButton("Jump Button: "
                            +
KeyEvent.getKeyText(GamePanel.jumpButton));
                  jump.addActionListener(this);
                  jump.setPreferredSize(buttonSize);
                  add(jump);
                  left = new GameButton("Left Button: "
                            +
KeyEvent.getKeyText(GamePanel.leftButton));
                  left.addActionListener(this);
                  left.setPreferredSize(buttonSize);
                  add(left);
                  right = new GameButton("Right Button: "
                            +
KeyEvent.getKeyText(GamePanel.rightButton));
                  right.addActionListener(this);
                  right.setPreferredSize(buttonSize);
                  add(right);
                  use = new GameButton("Interaction Button: "
                            +
KeyEvent.getKeyText(GamePanel.useButton));
                  use.addActionListener(this);
                  use.setPreferredSize(buttonSize);
                  add(use);
                  restart = new GameButton("Restart Button: "
                            +
KeyEvent.getKeyText(GamePanel.restartButton));
                  restart.addActionListener(this);
                  restart.setPreferredSize(buttonSize);
                  add(restart);

                  GameButton back = new GameButton("Back");
                  back.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                              navigateTo(new MainMenu());
                        }
                  });
                  back.setPreferredSize(buttonSize);
                  add(back);
            }

            public void resetButtons() {
                  jump.setText("Jump Button: "
                            +
KeyEvent.getKeyText(GamePanel.jumpButton));
                  left.setText("Left Button: "
                            +
KeyEvent.getKeyText(GamePanel.leftButton));
                  right.setText("Right Button: "
                            +
KeyEvent.getKeyText(GamePanel.rightButton));
```

```
                use.setText("Interaction Button: "
                            +
KeyEvent.getKeyText(GamePanel.useButton));
                restart.setText("Restart Button: "
                            +
KeyEvent.getKeyText(GamePanel.restartButton));
            }

            public void actionPerformed(ActionEvent e) {
                if (selected != null)
                    resetButtons();
                selected = (GameButton) (e.getSource());
                selected.setText("Press any key to set");
                requestFocus();
            }

            public void keyPressed(KeyEvent e) {
                if (selected != null) {
                    if (selected == jump)
                        GamePanel.jumpButton = e.getKeyCode();
                    else if (selected == left)
                        GamePanel.leftButton = e.getKeyCode();
                    else if (selected == right)
                        GamePanel.rightButton = e.getKeyCode();
                    else if (selected == use)
                        GamePanel.useButton = e.getKeyCode();
                    else if (selected == restart)
                        GamePanel.restartButton = e.getKeyCode();
                    GamePanel.saveKeyBindings();
                    selected = null;
                    resetButtons();
                }
            }

            public void keyReleased(KeyEvent e) {
            }

            public void keyTyped(KeyEvent e) {
            }
        }

        private class PlayOrEditActionListener implements ActionListener {
            private String filename;

            public PlayOrEditActionListener(String fileName) {
                this.filename = fileName;
            }

            public void actionPerformed(ActionEvent e) {
                File file = new File("levels/" + filename);
                GamePanel gp = new GamePanel();
                gp.setGameMenu(GameMenu.this);
                Game g = new Game(gp);
                gp.setGame(g);
                g.loadLevel(file);
                navigateTo(gp);
                gp.requestFocus();
                g.start();
            }
      }

      private class PlayMenu extends JPanel {
            private static final long serialVersionUID = -
7010447333195916816L;
            private final Dimension size = new Dimension(600, 500);
            private JList<String> list;
            private GameButton playCustom;

            public PlayMenu() {
                  setPreferredSize(size);
                  setBackground(new Color(0, 0, 0, 125));
                  setBorder(new EmptyBorder(new Insets(20, 20, 20, 20)));
                  setLayout(new BorderLayout());
                  JPanel panel = new JPanel();
                  panel.setLayout(new GridLayout(0, 2, 10, 10));
                  panel.setOpaque(false);
                  JPanel premade = new JPanel();
                  premade.setOpaque(false);
                  premade.setLayout(new GridLayout(3, 3, 10, 10));
                  premade.setBorder(new TitledBorder(new EmptyBorder(new
Insets(40,
                              20, 20, 20)), "Premade Levels",
                              TitledBorder.DEFAULT_JUSTIFICATION,
TitledBorder.TOP,
                              new Font("Calibri", Font.PLAIN, 16),
Color.white));
                  for (int i = 0; i < AMOUNT_OF_PREMADE_LEVELS; i++) {
                        premade.add(new GameButton("" + (i + 1)));
                        ((GameButton) premade.getComponent(i))
                                    .addActionListener(new
PlayOrEditActionListener(
                                                "premade/" + (i + 1) +
".level"));
                  }
                  JPanel custom = new JPanel();
                  custom.setBorder(new TitledBorder(new EmptyBorder(new
Insets(40,
                              20, 20, 20)), "Custom Levels",
                              TitledBorder.DEFAULT_JUSTIFICATION,
TitledBorder.TOP,
                              new Font("Calibri", Font.PLAIN, 16),
Color.white));
                  custom.setLayout(new BorderLayout(0, 10));
                  custom.setOpaque(false);
                  list = listCustoms();
                  custom.add(list, BorderLayout.CENTER);
                  playCustom = new GameButton("Play");
                  playCustom.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                              if (list.getSelectedValue() != null) {
                                    (new
PlayOrEditActionListener(list.getSelectedValue()))
                                                .actionPerformed(e);
                              }
                        }
                  });
```

```
			playCustom.setPreferredSize(new Dimension(100, 30));
			playCustom.setEnabled(false);
			list.addListSelectionListener(new
ListSelectionListener() {
				public void valueChanged(ListSelectionEvent e) {
					playCustom.setEnabled(true);
				}
			});
			custom.add(playCustom, "South");
			panel.add(premade);
			panel.add(custom);
			add(panel, "Center");
			GameButton backButton = new GameButton("Back");
			backButton.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					navigateTo(new MainMenu());
				}
			});
			backButton.setPreferredSize(new Dimension(100, 30));
			JPanel backPanel = new JPanel();
			backPanel.setOpaque(false);
			backPanel.add(backButton);
			add(backPanel, "South");
		}

		public JList<String> listCustoms() {
			File[] files = (new File("levels/"))
					.listFiles(new FilenameFilter() {
						public boolean accept(File dir, String
name) {
							return
name.toLowerCase().endsWith(".level");
						}
					});
			String[] names = new String[files.length];
			for (int i = 0; i < files.length; i++) {
				names[i] = files[i].getName();
			}
			JList<String> result = new JList<String>(names);
			result.setBorder(new EmptyBorder(new Insets(5, 5, 5,
5)));

	result.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
			return result;
		}

		@Override
		public Dimension getPreferredSize() {
			return size;
		}

		@Override
		public Dimension getMinimumSize() {
			return size;
		}
	}

	private class GameDialog extends JPanel {
		private static final long serialVersionUID = -
3244577989685663540L;

		protected final Dimension BUTTON_SIZE = new Dimension(150,
30);

		protected JPanel panel;

		public GameDialog(Dimension size) {
			setOpaque(false);
			setLayout(new GridBagLayout());
			setSize(size);
			panel = new JPanel();
			panel.setBackground(new Color(0, 0, 0, 175));
			panel.setLayout(new GridLayout(0, 1, 10, 10));
			panel.setBorder(new EmptyBorder(new Insets(10, 10, 10,
10)));
			add(panel);
		}

		public GameButton addButton(String text, ActionListener al)
{
			GameButton gb = new GameButton(text);
			gb.addActionListener(al);
			gb.setPreferredSize(BUTTON_SIZE);
			panel.add(gb);
			return gb;
		}

		@Override
		protected void paintComponent(Graphics g) {
			g.setColor(new Color(0, 0, 0, 100));
			g.fillRect(getLocation().x, getLocation().y,
getWidth(),
					getHeight());
		}
	}

	private class PauseMenu extends GameDialog {
		private static final long serialVersionUID =
3340418322179036264L;

		public PauseMenu(Dimension size) {
			super(size);
			addButton("Resume", new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					showHideGameDialog(new GameDialog(new
Dimension()));
				}
			});
			addButton("Exit", new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					navigateTo(new MainMenu());
				}
			});
		}
	}
	private class LevelCompleteMenu extends GameDialog {
		private static final long serialVersionUID = -
7873635041415097811L;

		private GamePanel gamePanel;
		private GameButton nextLevel;

		public LevelCompleteMenu(Dimension size, GamePanel gp) {
			super(size);
			gamePanel = gp;
			JLabel label = new JLabel("Level Complete!");
			label.setFont(new Font("Arial", Font.BOLD, 20));
			label.setForeground(Color.WHITE);
			panel.add(label);
			nextLevel = addButton("Next Level", new
ActionListener() {
				public void actionPerformed(ActionEvent e) {
					String filePath =
gamePanel.getGame().getLevel().getFile()
							.getAbsolutePath();
					int nextLevelNo = Integer.parseInt(""
							+
filePath.charAt(filePath.lastIndexOf('/') + 1)) + 1;
					(new PlayOrEditActionListener("premade/" +
nextLevelNo
							+ ".level")).actionPerformed(e);

				}
			});
			nextLevel.setVisible(false);
			addButton("Exit", new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					navigateTo(new PlayMenu());
				}
			});
		}

		public void setNextLevelEnabled(boolean b) {
			nextLevel.setVisible(b);
		}
	}

	private class PauseMenuWithRestart extends GameDialog {
		private static final long serialVersionUID =
3340418322179036264L;

		private GamePanel gamePanel;

		public PauseMenuWithRestart(Dimension size, GamePanel gp) {
			super(size);
			this.gamePanel = gp;
			addButton("Resume", new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					showHideGameDialog(new GameDialog(new
Dimension()));
				}
			});
			addButton("Restart", new ActionListener() {
```

```
                    public void actionPerformed(ActionEvent e) {
                         gamePanel.end(false);
                         showHideGameDialog(new GameDialog(new
Dimension()));
                    }
               });
               addButton("Exit", new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                         navigateTo(new PlayMenu());
                    }
               });
          }
     }

     private class CreditsPanel extends JPanel {
          private static final long serialVersionUID =
2449727556367804166L;

          public CreditsPanel() {
               setBackground(new Color(0, 0, 0, 125));
               setLayout(new BorderLayout(0, 20));
               setBorder(new EmptyBorder(new Insets(40, 40, 40, 40)));
               JTextArea credits = new JTextArea();
               credits.setText("Created by:\n" + "Ateþ Balcý\n"
                         + "Batuhan Berk Yaþar\n" + "Ahmet Emre
Danýþman\n"
                         + "Buket Depren\n" + "Ayþe Ýrem Güner");
               credits.setOpaque(false);
               credits.setFont(new FontUIResource("Calibri",
Font.ITALIC, 20));
               credits.setForeground(Color.white);
               credits.setEnabled(false);
               add(credits, "Center");
               GameButton backButton = new GameButton("Back");
               backButton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                         navigateTo(new MainMenu());
                    }
               });
               backButton.setPreferredSize(new Dimension(100, 30));
               JPanel backPanel = new JPanel();
               backPanel.setOpaque(false);
               backPanel.add(backButton);
               add(backPanel, "South");
          }
     }

     private class HelpPanel extends JPanel {
          private static final long serialVersionUID = -
5876124166860168922L;

          private JLabel help;

          public HelpPanel() {
               super();
               setLayout(new BorderLayout());
               help = new JLabel(new ImageIcon("img/help.png"));
               setBackground(new Color(0, 0, 0, 125));
```

```
            add(help, "Center");
            GameButton backButton = new GameButton("Back");
            backButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    navigateTo(new MainMenu());
                }
            });
            JPanel backPanel = new JPanel();
            backPanel.setOpaque(false);
            backPanel.add(backButton);
            add(backPanel, "South");
            backButton.setPreferredSize(new Dimension(100, 30));
            backPanel.add(backButton);
        }
    }

    @Override
    public void componentMoved(ComponentEvent e) {
    }

    @Override
    public void componentShown(ComponentEvent e) {
    }

    @Override
    public void componentHidden(ComponentEvent e) {
    }
}
```