Multiplayer Platform Game
Group 19

Martin Petterson
Oskar Kvist
Christoffer Ekeroth
Misael Berrios Salas

March 10, 2008

Contents

1 Introduction

The purpose of this document is to establish a framework for how the software Scary Monsters is to be developed. As such, this document contains descriptions of all the classes, test cases and interfaces to external libraries that are needed in the development of Scary Monsters. At a higher level, user interfaces, states and hardware constraints are also specified.

The intended audience of this document is the programmers, architects and testers involved in the development process.

Readers should familiarize themselves with the Scary Monsters Requirements Document and keep it close at hand, since it is referenced to many times in this document.

1.1 Terms

- *Game stages* are the playing fields of the game, similar to race tracks in racing.
- The *.Net Framework* is a software component that is a part of Microsoft Windows operating systems. It has a large library of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework. Since code written in C# is compiled into the Microsoft Intermediate Language it can be run on any platform that supports the .Net Framework it can be run on Xbox 360.
- *Platform games* are a genre of videogames where the player must navigate through a side-scrolling environment (usually presented in two dimensions), often having to jump past obstacles, traps or enemies.
- *Scrolling* is the act of sliding the game presentation of the screen.
- *Xbox 360* is a games console developed by Microsoft.
- *XNA* (XNA's Not Acronymed) is both a C# game programming library and an extension to Visual C# Express. Games created with XNA can run on both Windows Vista/XP and Xbox 360.

1.2 Summary

Introduction:
The section you're currently reading. Except for this summary it contains an introduction to this document as well as definitions of technical terms used throughout the document.

System Overview:
The System Overview section contains a brief overview and general description of the system architecture in Scary Monsters, as well as a more detailed architecture description.

Design Considerations:
The Design Considerations section details assumptions that have affected the design process, as well as a list of constraints imposed upon the implementation of the system.

Graphical User Interface:
The Graphical User Interface section provides an overview of the graphical user interface in Scary Monsters using descriptive text and mock-up illustrations.

Design Details:
The Design Details section contains detailed information on how the system is supposed to be divided up into classes. It contains detailed information on all of the classes in the system as well as higher-level information about class responsibilities and how classes interact.

Functional Test Cases:
The Functional Test Cases section defines a number of tests that can be carried out in order to verify that the system satisfies the requirements defined in the Requirements Document.

2 System Overview

2.1 General Description

Scary Monsters is a video game that lets multiple players compete in head-to-head battles on the Xbox 360 game console. One of the main characteristics of Scary Monsters, and video games in general, is that it incorporates a wide array of inputs and outputs—the game must have a real-time graphical representation, there are music and sound effects, input has to be received from multiple devices, etc. To provide this functionality, several subsystems are needed, many of which need to be able to communicate with each other. Furthermore, the game world is filled with various types of game objects that need to interact with each other in many different ways. Also, the game can be in one of several states at any given time, which affects the behavior of almost all subsystems. Thus, the major problems we needed to solve in the design of Scary Monsters were:

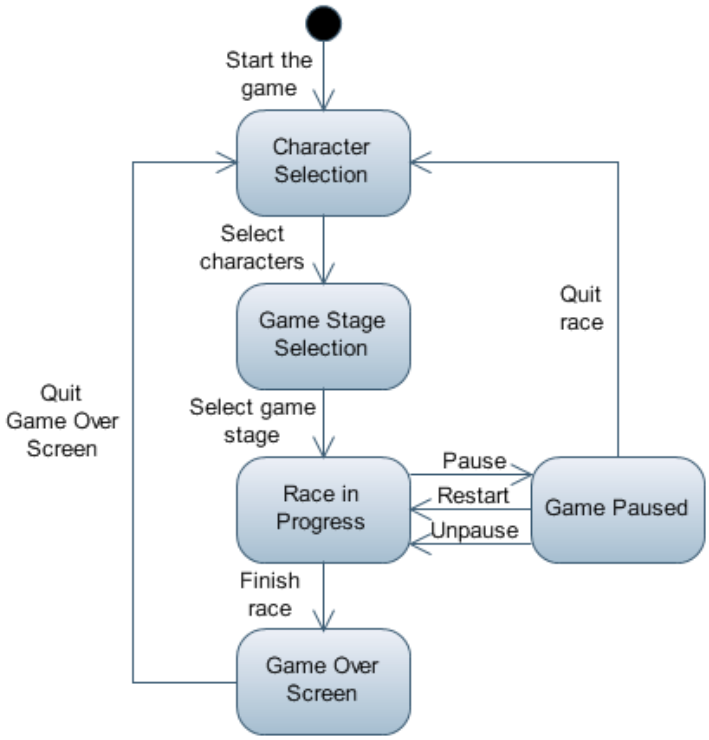
- Allowing subsystems to interface with each other in an elegant manner.
- Handling all the possible interactions between the different objects in the game world.
- Devising a system for managing and transitioning between the various states of the game.

The basic design approach is as follows:

- Responsibility for handling the different kinds of inputs and outputs needed in the game are divided across a number of subsystems, each of them responsible for handling one type of input/output.
- Responsibility for handling interactions between game objects is split up between the 2D physics engine Farseer and the game objects themselves. Farseer is responsible for detecting collisions between objects and notifying colliding objects of what they have collided with. The objects are in turn responsible for reacting to these collisions in the appropriate manner. In summary, Farseer keeps track of *who* interacts and *when*, and the different game objects keeps track of *how* they interact with each other.
- Responsibility for keeping track of game states is delegated to classes implementing the *IScreen* interface, each of which represents a game state.

2.2 Overall Architecture Description

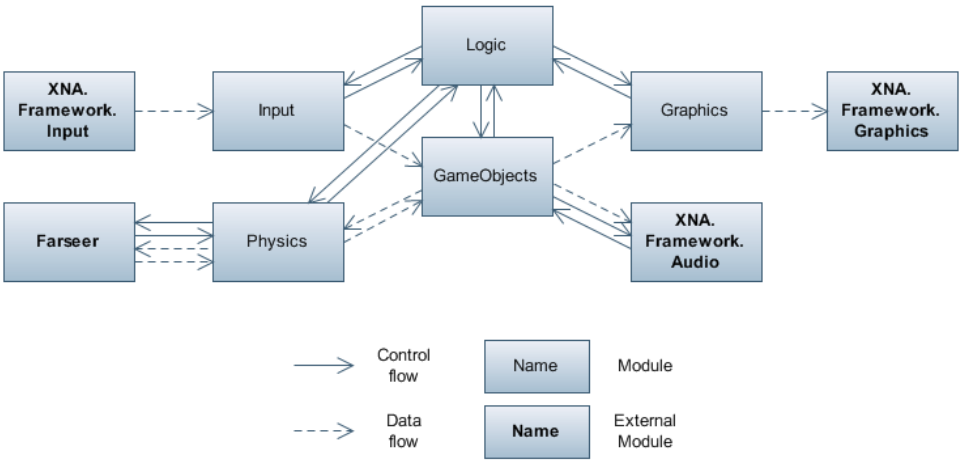
The following is a state transition diagram showing the states the game can be in as well as how the game can change state.



The game can be in any of five states at any given time. These states are:

- *Character Selection*—This is the initial menu presented to the players upon startup and before the start of each race. In order move on to the Game Stage Selection state, the players have to select characters.
- *Game Stage Selection*—In this state one of the players, whom is randomly chosen by the game, selects which game stage she wants to play on.
- *Race in Progress*—In this state the players play the actual game.
- *Game Paused*—In this state the race is paused and the players are presented with a menu. From here, the players can choose to quit the race, unpause the game or restart the race.
- *Game Over Screen*—In this state the players are presented with the positions they finished in and the times of their finishes and/or deaths.

Our main goal in designing the system was to divide the responsibilities of the system across modules in a logical and intuitive manner. The following diagram shows the modules the game is composed of as well as data and control flow between them.



- The *Logic module* contains the main game loop, and calls upon the other modules to perform tasks like receiving user input and updating the screen. The control flow therefore goes from the Logic module to the other modules and back again.

- The *Input module* is responsible for reading input from the players. The Input module reads the button states of connected controllers from the XNA Input module.

- The *Physics module* is responsible for calculating the positions and velocities for game objects as well as detecting collisions between them.

  The reason behind the division into Physics and Logic modules is twofold—one is that we want to be able to use a physics module developed by a third party, the other is that dividing responsibilities accross modules in this way makes each module smaller and more manageable.

- The *Graphics module* is responsible for drawing to the screen. The Graphics module reads data about the game world from the GameObjects repository and utilizes calls to the XNA Graphics module to draw the appropriate representations of the GameObjects to the screen. Each GameObject is responsible for keeping track of its graphical representation.

- The *GameObjects module* is a data repository containing information about game objects such as players, traps, monsters, platforms, etc.

  Note, however, that the Game Objects module it not purely a data repository. It contains objects that have methods associated with them.

The game runs in a continous loop, in all states of the game. In the Race in Progress state, the following things happen during every iteration of the loop:

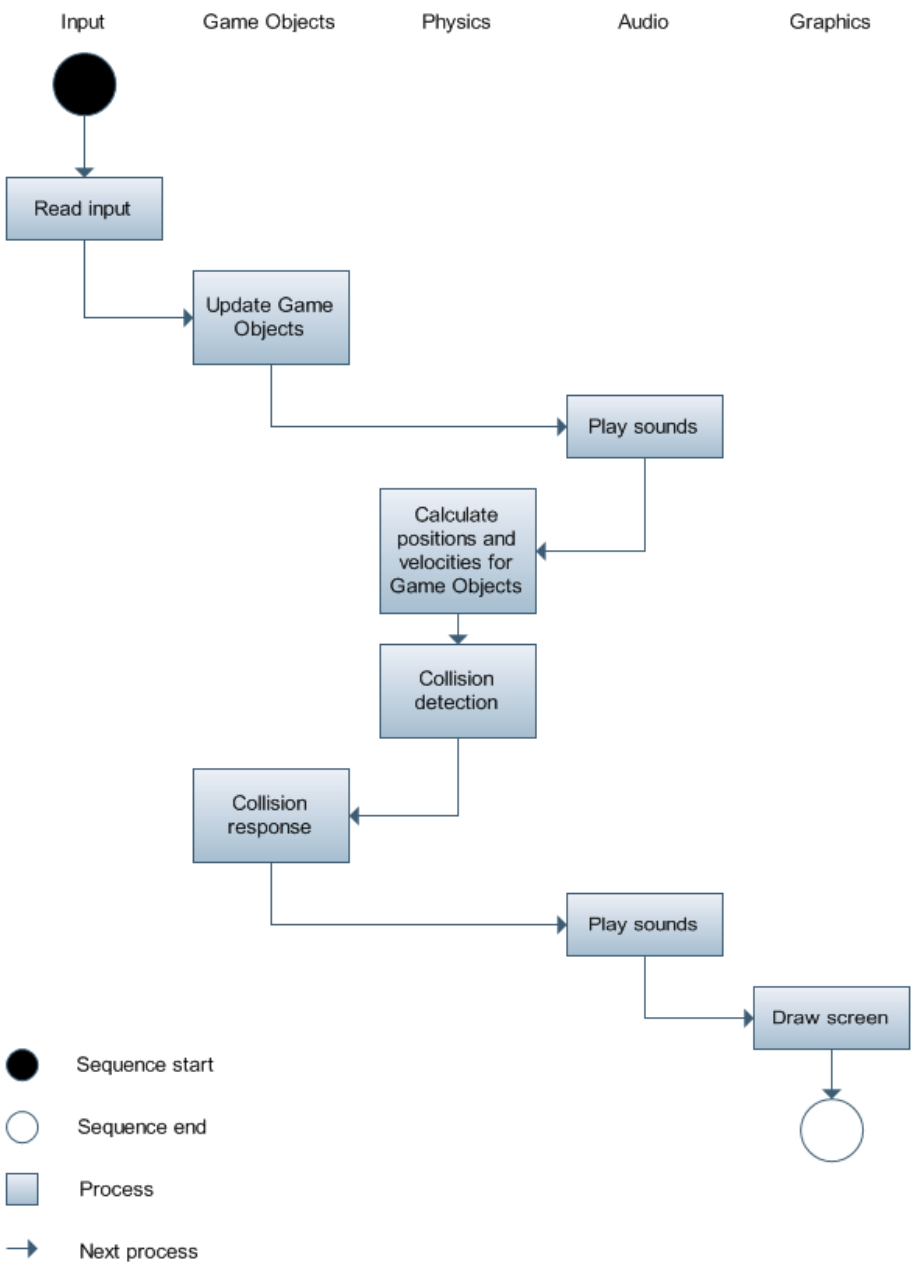
1. The Logic module tells the Input module to read input from the players.
2. The Logic module tells the game objects to update themselves. For player objects, this includes making use of the new input. These updates can cause the game to play sound effects.
3. The Logic module tells the Physics module to calculate positions and velocities for all game objects and to detect collisions between them. Collisions between game objects can cause the game to play sound effects.
4. The Logic module tells the Graphics module to update the graphics on the screen.

In all other states, the following things happen during every iteration of the loop:

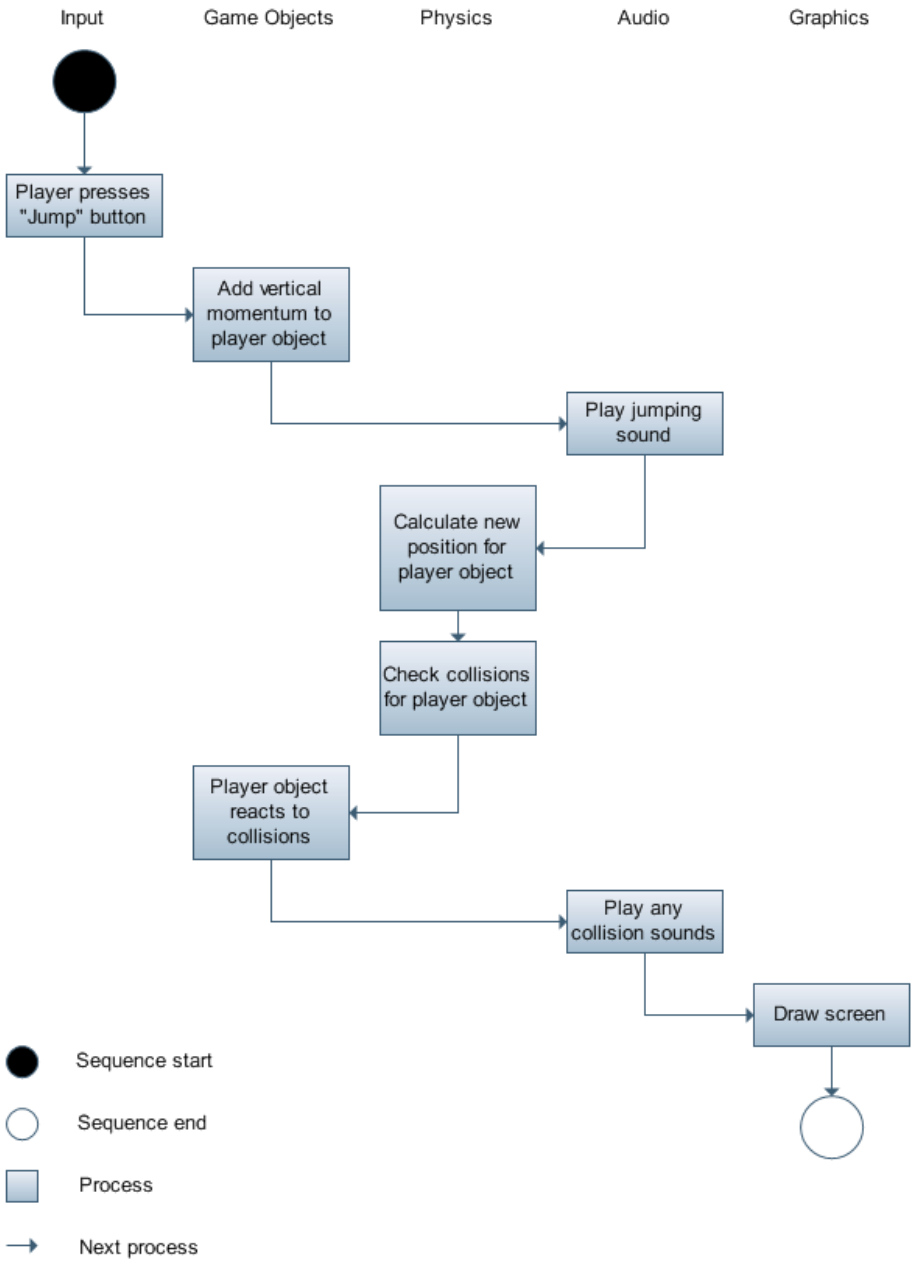
1. The Logic module tells the Input module to read input from the players.
2. The Logic module interprets the input. This may cause the game to play sound effects.
3. The Logic module tells the Graphics module to update the graphics on the screen.

2.3 Detailed Architecture

The following diagram shows data and order of execution during an iteration of the game loop in the general case. Recall that the Logic module calls upon the other modules during the game loop; the control flow goes from the Logic module and back again.



The following diagram shows data and order of execution during an iteration of the game loop in the case of a player jumping. Recall that the Logic module calls upon the other modules during the game loop; the control flow goes from the Logic module and back again.



3 Design Considerations

Most of the design considerations that will have to be made during development stem from the fact that the software will be developed for the Xbox 360 games console. Further considerations must also be made due to the fact that the game will use the XNA framework, which means it will run on the .NET Portable Framework present on the Xbox 360.

3.1 Assumptions and Dependencies

3.1.1 Related Software and Hardware

- Xbox 360 games console—The main hardware platform for Scary Monsters.
- Xbox 360 gamepads—The input devices used to play the game.
- XNA Framework—A game programming library that will be used in the development of Scary Monsters.
- .NET Portable Framework Class Library—A class library that will be used in the development of Scary Monsters.
- .NET Portable Framework—The environment in which the interpreted code will run on the Xbox 360.

3.1.2 End-user Characteristics

Our target end-user has previous experience with the Xbox 360 game console or other similar game systems, at least to the extent that she is familiar and comfortable with a gamepad as an input device. Except for being able to operate the Xbox 360 console, no further technical knowledge is required from the user.

3.1.3 Possible and/or Probable Changes in Functionality

The following changes in functionality will possibly or likely take place during the course of development:

- Changes to game-related constants, such as the running speed of the characters or how many lives each player start out with.
- Replacing the Farseer physics library with our own code in case integration of Farseer proves to be problematic.
- The addition of new types of game objects, such as new enemies or power-ups.

3.2 General Constraints

3.2.1 Xbox 360 Hardware

The Xbox 360 has 512 MB of RAM. However, not all of it is available to us at any one time since the .NET Portable Framework, Xbox 360 operating system and Xbox 360 dashboard need RAM of their own. The consequences of this is that we will definitely not be able to load more than 512 MB worths of data into memory and that we can never be sure exactly how much memory is available to us.

Due to the fact that XNA uses the .NET Portable Framework, Scary Monsters will not be able to use the processors on the Xbox 360 to their fullest potential. The nature of this constraint is described in detail below:

> (...) the Xbox 360 CPU cores have a couple of interesting problems associated with them. First, they have no out-of-order instruction fixup. This means that if the instructions arrive at the CPU in a way that isnt optimal, the CPU generates stalls to keep things in sync. This is in direct contrast to the Intel and AMD CPUs for Windows, which have a fair amount of logic to reorder instructions as they come in to help the CPU digest them. The Xbox 360 CPUs are also RISC-based processors. A complex instruction set processor like the Intel and AMD processors have many, many complex, essentially single-step instructions to do what will take 2 to 5 instructions to perform on a RISC processor. Since we are compiling this to MSIL and then Just-In-Time compiling it on the Xbox 360, the compiler doesn't have a lot of opportunity to reorder instructions the way the C++ compiler can. It also doesn't get much opportunity to make use of the tremendous numbers of registers on the Xbox 360 to prevent load and storing from memory to the actual operations. Since the C++ compiler is an offline compiler, it can crunch on the code and help the Xbox 360 processor by reordering instructions that would cause these stalls and also be "register aware" to prevent loads and stores. The MSIL compiler won't do this because it is processor agnostic, and the JITer is blissfully unaware of the CPU issues on the Xbox 360. The JITer must brute force everything.

The above is a quote from an article published on the XNA Creators Club homepage. The full article can be found at: `http://creators.xna.com/Downloads/GameStudio2/Optimization%20Tutorial%20-%20Particles%20and%20High-Frequency%20Code.doc`

The consequences of this is that if our code uses a lot of instruction jumps it might run unacceptably slow on the Xbox 360. It might become necessary to optimize the code by reducing the number of method calls.

3.2.2 The Living Room as the End-User Environment

A multitude of our possible user base will probably use our software in their living room, with their Xbox 360s connected to a television set. That means that the final output image from the Xbox360 might be downscaled to a low resolution (480 or 525 vertical lines) and / or viewed at a distance. This has several consequences, but the most important one is that small text might be hard to read in these conditions.

3.2.3 TV-Safe Areas

Older television sets are sometimes unable to display a complete picture, resulting in the edges of the picture being cut off. This means that important information, such as instructions to the players, should be kept within the "title-safe area" of the screen. For more information, see: `http://en.wikipedia.org/wiki/Safe_area`

3.2.4 The Xbox 360 Gamepad as an Input Device

The Xbox 360 gamepad has no keyboard by default, though it can be purchased as an accessory. Text input is possible without the keyboard, but can be frustrating and time-consuming and so it should be avoided.

3.2.5 Complying with Xbox 360 Interface Standards

In most Xbox 360 user interfaces, "green" means moving forwards in a menu and "red" means going backwards. Scary Monsters should ideally follow this standard, in order not to confuse players.

3.2.6 Performance Requirements

Responsiveness and a stable frame rate are more important in games than in many other applications. If the user can detect decreases in frame rate or feels that the game is unresponsive it will have a significant negative impact on the user experience.

3.2.7 Storage Devices on the Xbox 360

Most Xbox 360 consoles that are sold today come equipped with a hard drive, but there are older models that lack hard drives. Some form of storage device is required to play XNA games, however, and thus it is safe to assume that the user has at least a memory card. The size of the smallest memory card is 64 megabytes, and most likely the user has data on the memory card that she does not wish to delete to make room for Scary Monsters. Thus, any increase in the storage requirements for Scary Monsters will mean a decrease in the potential user base.

4 Graphical User Interface

4.1 User Interface Overview

The user interface is divided across five screens (see section 2.2 for a state diagram):

- Character Select Screen
- Game Stage Select Screen
- Game Screen
- Pause Screen
- Game Over Screen

The players are initially presented with the Character Selection Screen. At this screen each player confirms her participation in the race and selects a character.

When the players have confirmed they are done selecting characters they are taken to the Game Stage Selection Screen. The player who gets to choose what game stage to play on is chosen by the game at random. In order to start the race the chosen player selects one of the game stages.

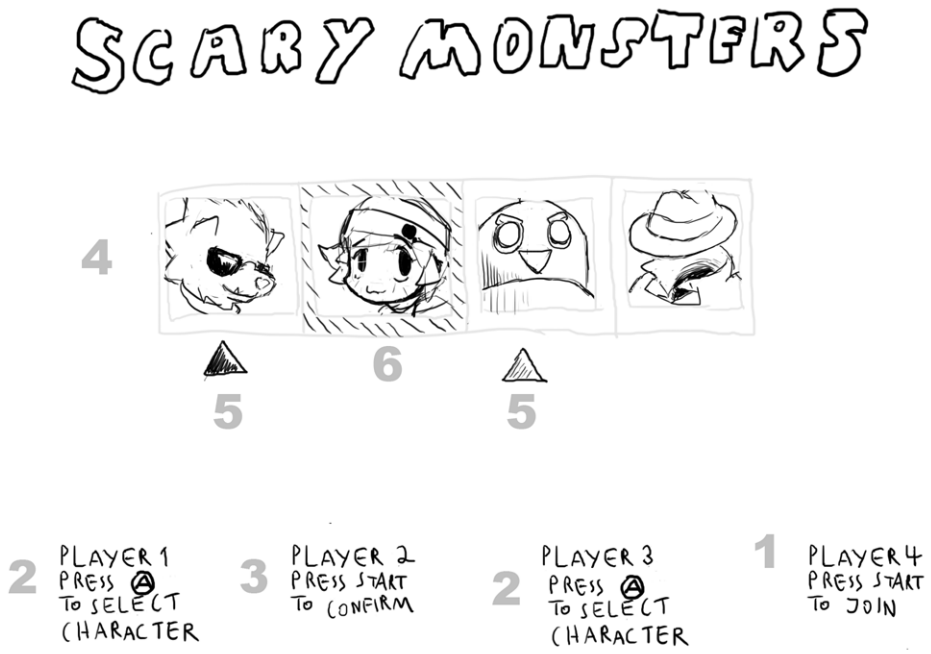
When the game stage is chosen the players are taken to the Game Running Screen. This is where the actual game is played.

At any time during the race, players may pause the game. They are then presented with the Pause Screen. From the Pause Screen they can either choose to resume the race, restart the race or exit to the Character Selection Screen.

After the race is finished the Game Over Screen is shown, where the players are presented with their finish and/or death times. When every player has confirmed that they are done viewing the results, the game returns to the Character Selection Screen.

4.2 Pictures

4.2.1 Character Selection Screen



This is the character selection screen. Players must first press the Start button in order to confirm their participation in the next race (1). Once a player has confirmed that she is to participate she can select one of the characters in the table in the middle of the screen (4). The players move selection arrows (5) with the analog sticks on their gamepads between the characters and confirm their choices by pressing the A button (2). A player may not choose the same character as someone else. If a character has been selected, its table cell turns grey (6). A player can undo her choice of character by pressing the B button. The arrows have different colors to make them distinguishable from each other. Once a player has selected a character she confirms that she is ready to start the race by pressing the Start button (3). When every participating player has done so they are taken to the game stage selection screen.

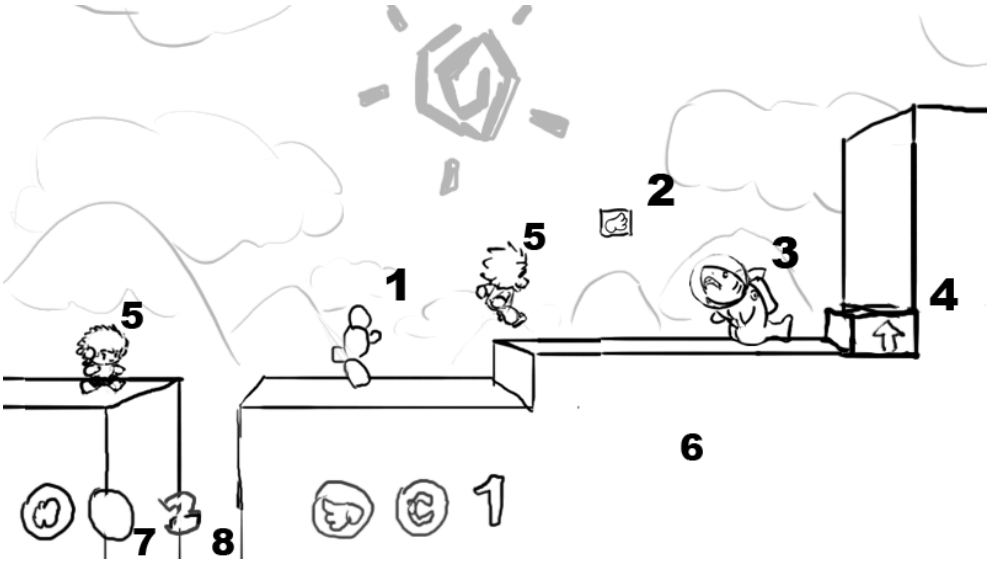
This screen fulfills the functional requirement that the players shall be able to select characters (Requirements Document section 4.1.2, requirement 8(a)).

4.2.2 Game Stage Selection Screen

This is the Game Stage Selection Screen. A player is chosen, at random by the game, to choose a game stage (in this case player 3). While the chosen player browses the available game stages, the background changes to show a preview of the game stage currently pointed at by the arrow.

This screen fulfills the functional requirement that a randomly chosen player shall be able to select a game stage (Requirements Document section 4.1.2, requirements 8(d) and 8(e)).

4.2.3 Game Screen



This is the game screen, where the actual game is played. The game elements (see Requirements Document section 4.1.2, requirements 1(b), 1(e) and 2(a)) shown here are:

1. A trap (a stationary cactus)
2. A power-up dispenser (currently showing the levitation power-up icon)
3. A monster
4. A springboard
5. Players
6. A platform

This screen also shows the head-up display. Each player is shown her two power-up slots (7) and how many lives she has left (8). Player one has her head-up display to the far left of the screen. Player two has her head-up display to the right of player one's, and so on, with player four having her head-up display to the far right. In the picture above, only player one and two are playing. (See Requirements Document section 4.1.2, requirement 5(a))

Notice also that the screen is not split up into smaller areas for each player. Instead the players all use the whole screen when playing. (See Requirements Document section 4.1.2, requirement 5(b))

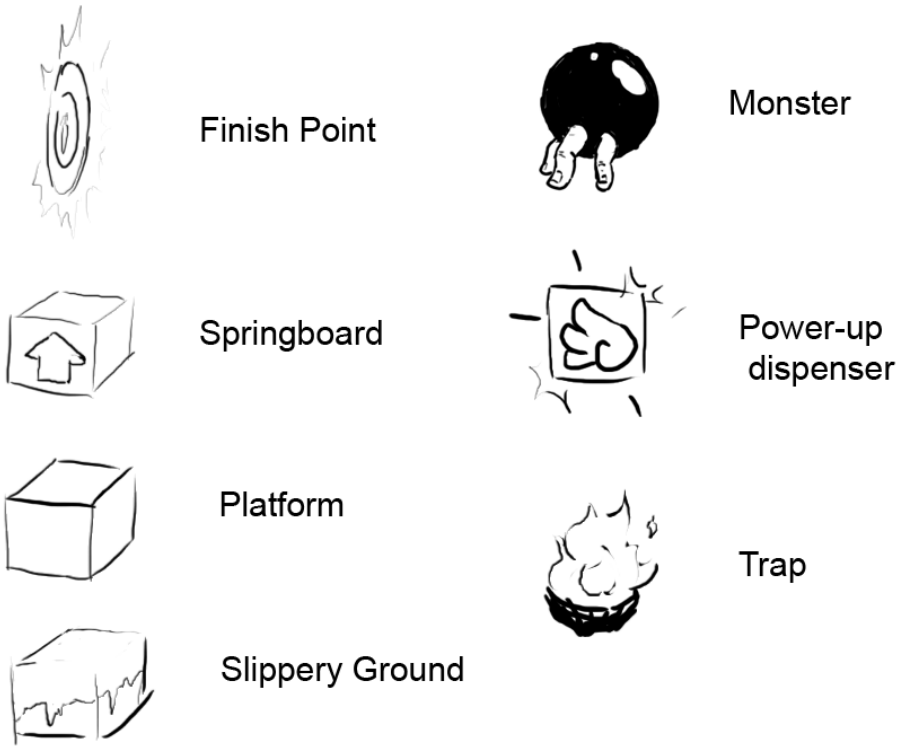
4.2.4 Pause Screen

This is the Pause Screen. While a race is in progress, any player may press the Start button on her gamepad to pause the game. When the game is paused, everything in the race freezes and the players are presented with the pause menu. Only the player that paused the game may navigate the pause menu. (See Requirements Document section 4.1.2, requirement 9)

4.2.5 Game Over Screen

This is the Game Over Screen. When all players have either reached the finish point or lost all of their lives, the race ends and the Game Over Screen is shown. Here, the players are presented with their finish times and/or death times. (See Requirements Document section 4.1.2, requirement 10(a))

4.2.6 Game Stage Elements



The picure above shows the game stage elements as described in the Requirements Document section 4.1.2, requirement 2 with subitems.

4.2.7 Power-up Icons

These are the power-up icons. They are shown both on the power-up dispensers and as part of the head-up display (see section 4.2.3).

These icons correspond to the power-ups described in the Requirements Document section 4.1.2, requirement 4(d).

5 Design Details

5.1 CRC cards

5.1.1 ScaryMonsters

Responsibilities:

- To start up the game and keep it running until the user wants to quit.
- To instantite and store the instances of Logic, Physics, Input, Graphics, GameObjects and Camera.

Collaborators:

- CharacterSelectionScreen
- GameStageSelectionScreen
- GameRunningScreen
- PauseScreen
- GameOverScreen

5.1.2 Logic

Responsibilities:

- Enforcing game rules.

Collaborators:

- GameObjects

5.1.3 Input

Responsibilities:

- To instantiate and store four Controllers.
- To provide methods that abstract details of the input mechanism.

Collaborators:

- Controller

5.1.4 Physics

Responsibilities:

- To update the game objects' positions and velocities.
- To detect collisions between game objects.

Collaborators:

- GameObjects
- Geometry (Farseer)

5.1.5 Graphics

Responsibilities:

- To draw everything

Collaborators:

- GraphicsDevice (XNA)
- GameObjects
- SpriteObject
- ModelObject
- Camera

5.1.6 GameObjects

Responsibilities:

- To load GameStages from files and store them.
- To store all instances of subclasses GameObject in the game.

Collaborators:

- GameStage
- Player
- Trap
- MovingPlatform
- Monster
- SpringBoard

- BoxingGlove
- Claw
- ShrinkingRay
- BananaPeel
- Fetter
- StartingPoint
- FinishPoint
- PowerupDispenser

5.1.7 Camera

Responsibilities:

- To store the camera position.
- To update the camera position properly as the game progresses.

Collaborators:

- GameObjects
- Player

5.1.8 GameStage

Responsibilities:

- To represent a Game Stage; keeping track of platforms, finish point and start points.

Collaborators:

- StartPoint
- FinishPoint

5.1.9 GameObject (abstract class, extends Geometry)

Responsibilities:

- To store the position, rotation and physical model of an object in the game.

No collaborators.

5.1.10 SpriteObject (abstract class, extends GameObject)

Responsibilities:

- To store the graphical representation (a sprite) of a GameObject.

No collaborators.

5.1.11 ModelObject (abstract class, extends GameObject)

Responsibilities:

- To store the graphical representation (a model) of a GameObject.

No collaborators.

5.1.12 Controller

Responsibilities:

- Represents a game pad device.
- Abstract details that is not important to the game.

Collaborators:

- GamePad (XNA)

5.1.13 Path

Responsibilities:

- To represent a path that a MovingPlatform or Trap follows.

No collaborators.

5.1.14 Effect (abstract class)

Responsibilities:

- Provide shared functionality for all Effects.

No collaborators.

5.1.15 StunEffect (extends Effect)

Responsibilities:

- Provides functionality associated with stun effects.

Collaborators:

- Player
- Monster

5.1.16 ShrinkEffect (extends Effect)

Responsibilities:

- Provides functionality associated with the Shrinking Ray power-up.

Collaborators:

- Player
- Monster

5.1.17 SpeedEffect (extends Effect)

Responsibilities:

- Provides functionality associated with the Speed Boost power-up.

Collaborators:

- Player

5.1.18 LevitationEffect (extends Effect)

Responsibilities:

- Provides functionality associated with the Levitation power-up.

Collaborators:

- Player

5.1.19 InvulnerabilityEffect (extends Effect)

Responsibilities:

- Provides functionality associated with invulnerability effects.

Collaborators:

- Player

5.1.20 FetterEffect (extends Effect)

Responsibilities:

- Provides functionality associated with the Fetter power-up.

Collaborators:

- Player
- Monster

5.1.21 ClawEffect (extends Effect)

Responsibilities:

- Provides functionality associated with the Claw power-up.

Collaborators:

- Player
- Monster

5.1.22 SlipEffect (extends Effect)

Responsibilities:

- Provides functionality associated with the Banana Peels power-up and with slippery ground.

Collaborators:

- Player
- Monster

5.1.23 Player (extends ModelObject)

Responsibilities:

- To represent player.
- To update the player—getting input from a Controller and updating the Effects it is affected by.

Collaborators:

- Controller
- Effect

5.1.24 Monster (extends ModelObject)

Responsibilities:

- To represent a monster.
- To update the monster—updating the movement of the monster and updating the Effects it is affected by.

Collaborators:

- Effect

5.1.25 MovingPlatform (extends GameObject)

Responsibilities:

- To represent a moving platform; keeping track of position and path.

Collaborators:

- Path

5.1.26 PowerupDispenser (extends SpriteObject)

Responsibilities:

- Represents a power-up dispenser.

No collaborators.

5.1.27 FinishPoint (extends SpriteObject)

Responsibilities:

- Represents the Finish Point

No collaborators.

5.1.28 StartingPoint (extends GameObject)

Responsibilities:

- Represents a Starting Point

No collaborators.

5.1.29 Springboard (extends ModelObject)

Responsibilities:

- Represents a Springboard

No collaborators.

5.1.30 Trap (extends ModelObject)

Responsibilities:

- Represents a trap.

No collaborators.

5.1.31 BoxingGlove (extends SpriteObject)

Responsibilities:

- Represents a boxing glove that is spawned when the Boxing Glove Power-up is applied.

No collaborators.

5.1.32 Fetter (extends SpriteObject)

Responsibilities:

- Represents a fetter that is spawned when the Fetter Power-up is applied.

No collaborators.

5.1.33 ShrinkingRay (extends SpriteObject)

Responsibilities:

- Represents a shrinking ray that is spawned when the Shrinking Ray Power-up is applied.

No collaborators.

5.1.34 Claw (extends SpriteObject)

Responsibilities:

- Represents a claw that is spawned when the Claw Power-up is applied.

No collaborators.

5.1.35 BananaPeel (extends SpriteObject)

Responsibilities:

- Represents a banana peel that is spawned when the Banana Peels Power-up is applied.

No collaborators.

5.1.36 CharacterSelectionScreen

Responsibilities:

- Represent the state in which the players select characters.
- Keep track of which players are in the game and what characters they have selected.

Collaborators:

- Controller
- Input
- Graphics

5.1.37 GameStageSelectionScreen

Responsibilities:

- Represent the Game Stage select state.
- Keep track of who gets to select the game stage.
- Keep track of what game stage the player selects.

Collaborators:

- Menu
- MenuItem
- Controller
- Input

5.1.38 GameRunningState

Responsibilities:

- Represent the Game Running State (i.e. a race).
- Keep track of everything that happens within the game, through the use of the collaborators.

Collaborators:

- Logic
- Controller

- Input
- Physics
- Graphics
- GameObjects
- Camera
- HUD
- PauseScreen

5.1.39 PauseScreen

Responsibilities:

- Represent the Pause State.
- Keep track of who has paused the game, and what she selects.

Collaborators:

- Controller
- Menu
- MenuItem

5.1.40 GameOverScreen

Responsibilities:

- Represent the Game Over State.
- Shows finishing times.
- Keeps track of if the players have pressed Start or not, in effect keeping track of when to leave this state and go back to the character selection state.

Collaborators:

- Controller

5.1.41 Menu

Responsibilities:

- Represent a menu.
- Show menu items.
- Keeps track of what item the user selects.

Collaborators:

- MenuItem
- Controller

5.1.42 MenuItem

Responsibilities:

- Represent a menu item, consisting of text and images.

No Collaborators.

5.1.43 HUD

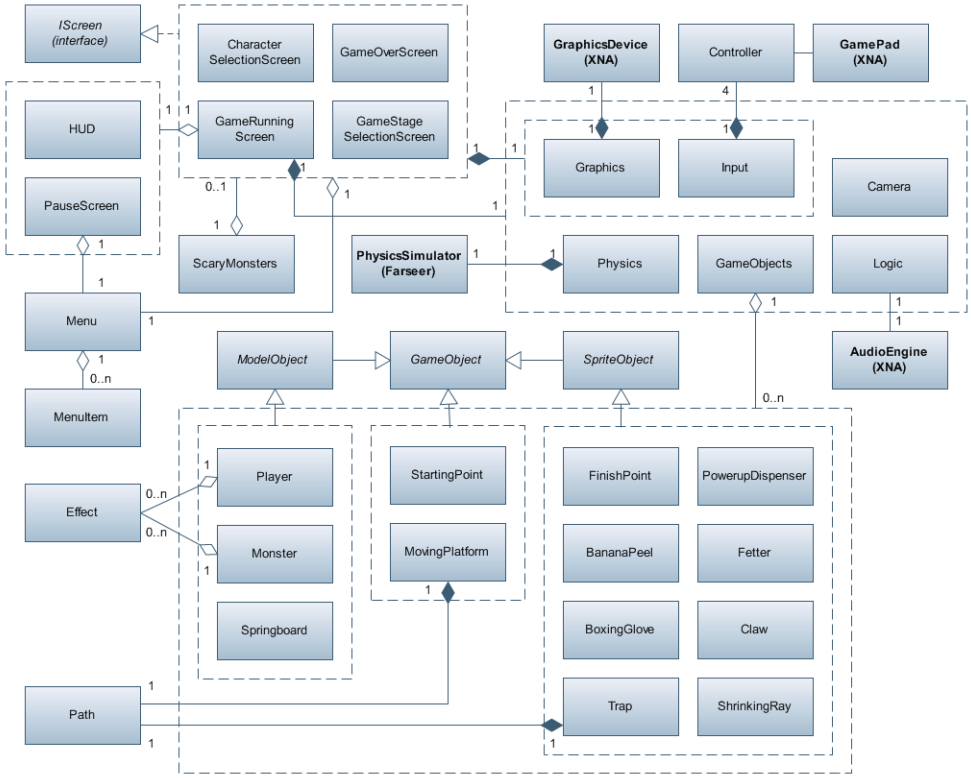
Responsibilities:

- Represent the HUD during a race.
- Show the players' current number of lives left and currently available power-ups.

Collaborators:

- Player

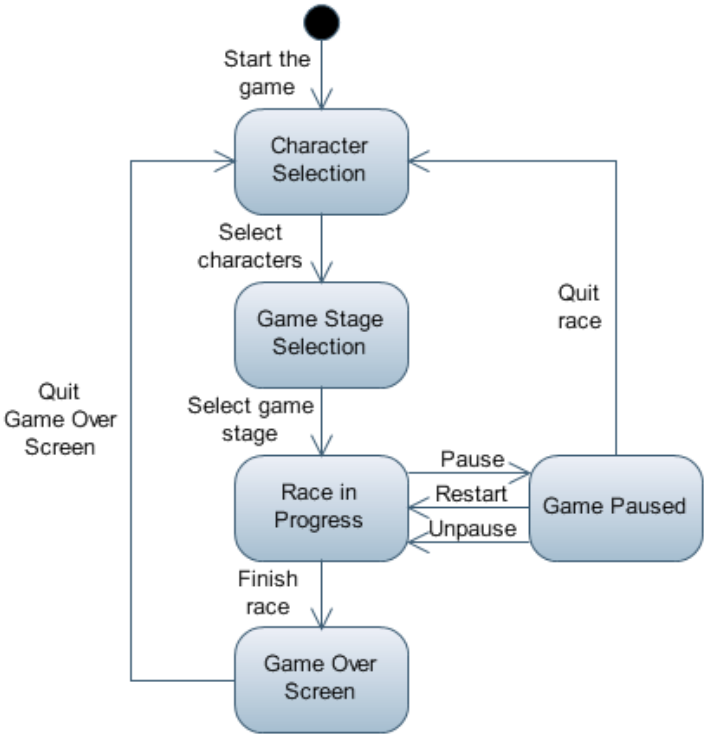
5.2 Class Diagram



5.3 State Charts

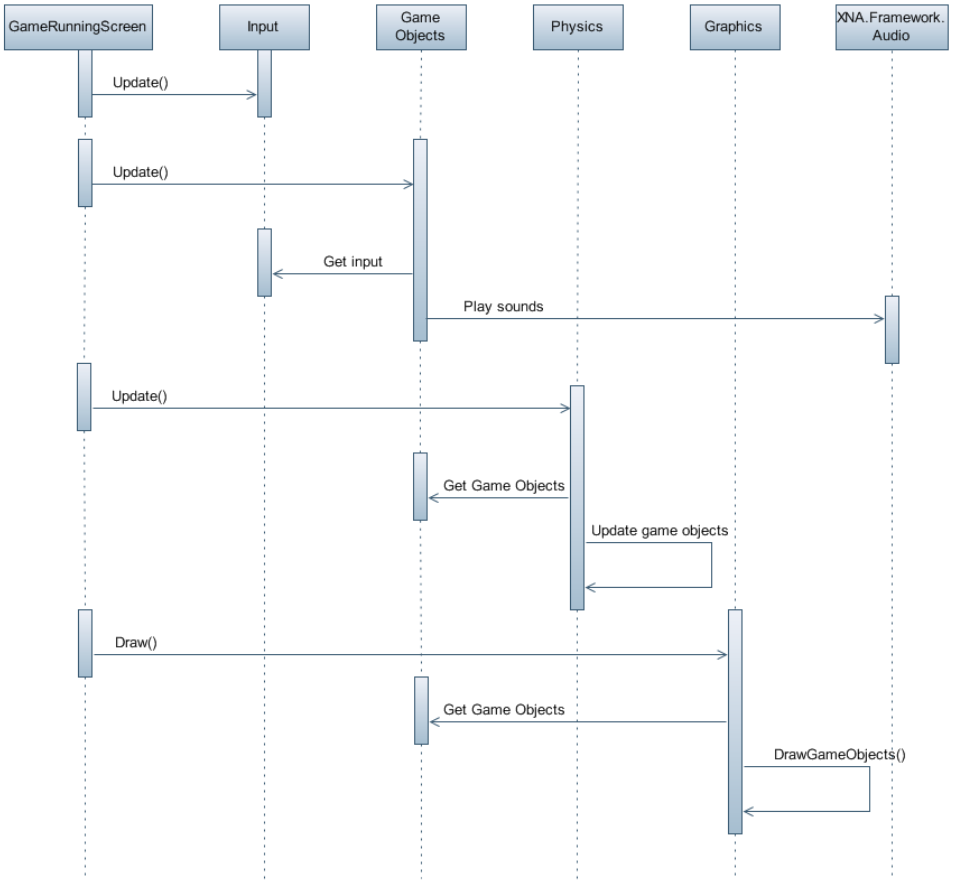
5.3.1 Overall system

See section 2.2 for a description of the state chart.



5.4 Interaction Diagrams

The following diagram shows the processes involved in the main game loop.



5.5 Detailed Design

5.5.1 ScaryMonsters

The ScaryMonsters class extends the Game class in XNA (Microsoft.Xna.Framework.Game).
ScaryMonsters is responsible for starting up the game. It is also responsible for updating and drawing the different screens in the game. The currently active screen is stored in the activeScreen field.

Fields:

- private GraphicsDeviceManager device
- private IScreen activeScreen

Properties:

- public IScreen ActiveScreen

Methods:

- ScaryMonsters()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: The constructor of the ScaryMonsters class. Instantiates the XNA class GraphicsDeviceManager.
  Data it accesses: None
  Preconditions: None
  Postconditions: The GraphicsDeviceManager has been instantiated.
  Called by: Main()
  Calls: None

- LoadContent()

  Access modifier: protected
  Parameters: None
  Return value: None
  Description: Instantiates a CharacterSelectScreen and sets it as the currently active screen.
  Data it accesses: None
  Preconditions: None
  Postconditions: A CharacterSelectionScreen is made the currently active screen.
  Called by: Game.Run()
  Calls: CharacterSelectionScreen()

- Update()

  Access modifier: protected
  Parameters: None
  Return value: None
  Description: This method calls the Update() methods on the currently active screen.
  Data it accesses: None
  Preconditions: this.Initialize() must have been ran.
  Postconditions: Update() has been called on the active screen.
  Called by: Game.Run()
  Calls: activeScreen.Update()

- Draw()

  Access modifier: protected
  Parameters: None

Return value: None
Description: Calls the Draw() method on the currently active screen.
Data it accesses: None
Preconditions: this.LoadContent() must have been ran.
Postconditions: None
Called by: Game.Run()
Calls: activeScreen.Draw()

5.5.2 IScreen (interface)

IScreen is an interface that defines what methods a class needs to implement in order to function as a "screen".

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates this screen.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: ScaryMonsters.Update()
  Calls: Varies, because this is an interface.

- Draw()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws this screen.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: ScaryMonsters.Update()
  Calls: Graphics.Draw(IScreen)

5.5.3 CharacterSelectionScreen

The CharacterSelectionScreen class implements the IScreen interface. It represents the character selection screen in the game. It is responsible for providing a menu that lets players select characters.

It is also responsible for instantiating the GameStageSelectionScreen class and changing the ActiveScreen property in the ScaryMonsters instance, when the players are done selecting characters. Finally it is responsible for

providing the GameStageSelectionScreen class with data on what player selected which character.

The CharacterSelectionScreen class implements requirement 8(a), section 4.1.2 in the Requirements Document.
Fields:

- private ScaryMonsters game
- private Graphics graphics
- private Input input

Properties:

- public ScaryMonsters Game
- public Graphics Graphics
- public Input Input

Methods:

- CharacterSelectionScreen(ScaryMonsters game)

  Access modifier: public
  Parameters:
  ScaryMonsters game - the ScaryMonsters instance
  Return value: None
  Description: The constructor of the CharacterSelectionScreen class. Instantiates Graphics and Input and stores a reference to each instance in the fields.
  Data it accesses: None
  Preconditions: None
  Postconditions: Graphics and Input have both been instantiated and the instances stored in the fields of this instance of CharacterSelectionScreen.
  Called by: ScaryMonsters.LoadContent()
  Calls: None

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Calls Update() on input.

  If during the Update method the class notices that the players are done selecting characters, this method instantiates the GameStageSelectionScreen class and sets the instance as the activeScreen on the

ScaryMonsters instance *game*.
Data it accesses: None
Preconditions: None
Postconditions: Update() has been called on input.
Called by: ScaryMonsters.Update()
Calls: Input.Update(), GameStageSelectionScreen()

- Draw()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the CharacterSelectionScreen by calling Draw(this) on graphics.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: ScaryMonsters.Draw()
  Calls: Graphics.Draw(IScreen)

5.5.4 GameStageSelectionScreen

The GameStageSelectionScreen class implements the IScreen interface. It represents the game stage selection screen in the game. It is responsible for providing a menu that lets players select which game stage to play on. It is also responsible for instantiating the GameRunningScreen class and making it the currently active screen.

The GameStageSelectionScreen class implements requirement 8(d), section 4.1.2 in the Requirements Document.

Fields:

- private ScaryMonsters game
- private Graphics graphics
- private Input input
- private Controller[] controllers
- private int[] characters

Properties:

- public ScaryMonsters Game
- public Graphics Graphics
- public Input Input

Methods:

- GameStageSelectionScreen(ScaryMonsters game, Controller[] controllers, int[] characters)

  Access modifier: public
  Parameters:
  ScaryMonsters game - the ScaryMonsters instance
  Controller[] controllers - an array of size 4 with the Controllers of the players that are going to participate in the upcoming race. int[] characters - an array of size 4 with integers representing the characters that was chosen by the players during character selection.
  Return value: None
  Description: The constructor of the GameStageSelectionScreen class. Instantiates Graphics and Input and stores a reference to each instance in the fields.

  The controllers and characters parameters are saved for when this class instantiates the GameRunningClass.
  Data it accesses: None
  Preconditions: None
  Postconditions: Input and Graphics have been instantiated and the instances stored in the fields of this instance of GameStageSelectionScreen.
  Called by: ScaryMonsters.LoadContent()
  Calls: None

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Calls Update() on input.

  Also, if a game stage has been selected, this method is responsible for instantiating the GameRunningScreen class and setting it as the active screen in the ScaryMonsters instance *game*.
  Data it accesses: None
  Preconditions: None
  Postconditions: Update() has been called on input.
  Called by: ScaryMonsters.Update()
  Calls: Input.Update(), GameRunningScreen()

- Draw()

  Access modifier: public
  Parameters: None
  Return value: None

Description: Draws the GameStageSelectionScreen by calling Draw(this) on graphics.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by: ScaryMonsters.Draw()
Calls: Graphics.Draw(IScreen)

5.5.5 GameRunningScreen

The GameRunningScreen class implements the IScreen interface. It represents the race state (the actual game).

The GameRunningScreen class implements requirement 9, section 4.1.2 in the Requirements Document.

Fields:

- private ScaryMonsters game
- private Logic logic
- private Physics physics
- private Graphics graphics
- private Input input
- private GameObjects gameObjects
- private Camera camera

Properties:

- public ScaryMonsters Game
- public Logic Logic
- public Physics Physics
- public Graphics Graphics
- public Input Input
- public GameObjects GameObjects
- public Camera Camera

Methods:

- GameRunningScreen(ScaryMonsters game, Controller[] controllers, int[] characters, int gameStage)

  Access modifier: public
  Parameters:
  ScaryMonsters game - the ScaryMonsters instance
  Controller[] controllers - the controllers that the participating players use
  int[] characters - integers that tell which characters the players have chosen to play as
  int gameStage - an integer that tells which game stage to play on
  Return value: None
  Description: The constructor of the GameRunningScreen class. Instantiates Logic, Physics, Graphics, Input, GameObjects and Camera and stores a reference to each instance in the fields.

  When the GameObjects class is instantiated, the parameters *controllers*, *characters* and *gameStage* is used to create the correct characters and instantiate the correct GameStage.
  Data it accesses: None
  Preconditions: None
  Postconditions: Logic, Physics, Graphics, Audio, Input, GameObjects and Camera have all been instantiated and the instances stored in the fields of this instance of GameRunningScreen.
  Called by: GameStageSelectionScreen.Update()
  Calls: None

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Calls Update() on logic, physics, input, gameObjects and camera.

  Also, if the race is over, this method instantiates the GameOverScreen class, passing informating about finish times and death times to the GameOverScreen constructor.
  Data it accesses: None
  Preconditions: None
  Postconditions: Update() has been called on logic, physics, input, gameObjects and camera.
  Called by: ScaryMonsters.Update()
  Calls: Logic.Update(), Physics.Update(), Input.Update(), GameObjects.Update(), Camera.Update(), GameOverScreen()

- Draw()

Access modifier: public
Parameters: None
Return value: None
Description: Draws the GameRunningScreen by calling Draw(this) on graphics.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by: ScaryMonsters.Draw()
Calls: Graphics.Draw(IScreen)

5.5.6 GameOverScreen

The GameOverScreen class implements the IScreen interface. It represents the game over screen in the game. It is responsible for providing information to the players about their finishing and/or death times.

It is also responsible for instantiating the CharacterSelectionScreen class and making it the active screen in the ScaryMonsters instance *game*, once the players are done viewing the results of the last race.

The GameOverScreen class implements requirement 10(a), section 4.1.2 in the Requirements Document.

Fields:

- private ScaryMonsters game
- private Graphics graphics
- private Input input

Properties:

- public ScaryMonsters Game
- public Graphics Graphics
- public Input Input

Methods:

- GameOverScreen(ScaryMonsters game, bool[] finished, int[] times)

  Access modifier: public
  Parameters:
  ScaryMonsters game - the ScaryMonsters instance
  bool[] finished - an array with four slots, where true means that the player associated with that slot finished the race, and false means that the player died.

int[] times - an array with four slots containing the finish times and/or death times for each player, measured in ms. If the time is 0 in some slot, it means that the player associated with the slot did not participate.
Return value: None
Description: The constructor of the GameOverScreen class. Instantiates Graphics and Input and stores a reference to each instance in the fields.
Data it accesses: None
Preconditions: None
Postconditions: Input and Graphics have been instantiated and the instances stored in the fields of this instance of GameStageSelectionScreen.
Called by: GameRunningScreen.Update()
Calls: None

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Calls Update() on input.

  Also, if the players have signified that they are done viewing the results of the race, this method instantiates the CharacterSelectionScreen class, and makes it the active screen in the ScaryMonsters instance *game*.
  Data it accesses: None
  Preconditions: None
  Postconditions: Update() has been called on input.
  Called by: ScaryMonsters.Update()
  Calls: Input.Update(), CharacterSelectionScreen()

- Draw()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the GameOverScreen by calling Draw(this) on graphics.
  Data it accesses: None
  Preconditions: None
  Postconditions:
  Called by: ScaryMonsters.Draw()
  Calls: Graphics.Draw(IScreen)

5.5.7 Menu

A Menu represents a menu in the game. Each menu has several options that are represented by MenuItem objects.

Fields:

- private List<MenuItem> menuItems
- private MenuItem currentItem
- private bool done
- private Texture2D arrowGraphic

No properties

Methods:

- Menu()

  Access modifier: public
  Parameters:
  ScaryMonsters game - the ScaryMonsters instance
  Return value: None
  Description: The constructor of the Menu class. Instantiates a List of MenuItems.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: GameRunningScreen.GameRunningScreen(), CharacterSelectionScreen.CharacterSelectionScreen(), GameStageSelectionScreen.GameStageSelectionScre
  GameOverScreen.GameOverScreen()
  Calls: None

- AddMenuItem(MenuItem item)

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Adds a MenuItem to this Menu
  Data it accesses: None
  Preconditions: None
  Postconditions: item has been added to menuItems.
  Called by: GameRunningScreen.GameRunningScreen(), CharacterSelectionScreen.CharacterSelectionScreen(), GameStageSelectionScreen.GameStageSelectionScre
  GameOverScreen.GameOverScreen()
  Calls: None

- GetSelectedItem()

  Access modifier: public
  Parameters: None
  Return value: A MenuItem representing the item that was selected in the menu or null
  Description: The getSelectedItem method returns the MenuItem that was selected in the menu (currentItem) or null in case the user hasn't confirmed her selection yet (done is still set to null)
  Data it accesses: currentItem
  Preconditions: None
  Postconditions: The currently selected MenuItem has been returned
  Called by: GameRunningScreen.Update(), CharacterSelectionScreen.Update(), GameStageSelectionScreen.Update(), GameOverScreen.Update()
  Calls: None

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Checks wheter the stick on the controller has been moved or the A button has been pushed. If the stick has been moved, iterate one step forwards or backwards through the menuItems list depending on if the stick was moved downwards or upwards. If the A button has been pushed, set done to true.
  Data it accesses: Gains access to player input data from the IScreen.Input property
  Preconditions: None
  Postconditions: None.
  Called by: GameRunningScreen.Update(), CharacterSelectionScreen.Update(), GameStageSelectionScreen.Update(), GameOverScreen.Update()
  Calls: None

- Draw()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the Menu and its MenuItems. An arrow is drawn next to the currently selected menu item (currentItem) to indicate which item is currently selected.
  Data it accesses: None
  Preconditions: None
  Postconditions: The menu screen has been drawn.
  Called by: ScaryMonsters.Draw()
  Calls: Graphics.Draw(IScreen)

5.5.8 MenuItem

Fields:

- private String label
- private int value
- private Vector2 position

Properties:

- public String Label
- public int Value
- public Vector2 Position

Methods:

- MenuItem(String label, int value, Vector2 position) **Access modifier:** public
  Parameters:
  String label—The text that will be used to draw the graphical representation of this menu item.
  int value—An integer representing this menu item. These should map to constants defined in the class instantiating the menu item.
  Vector2 position—A 2D vector representing where on the screen the menu item should be drawn

  Return value: None
  Description: The constructor of the MenuItem class.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: GameRunningScreen.GameRunningScreen(), CharacterSelectionScreen.CharacterSelectionScreen(), GameStageSelectionScreen.GameStageSelectionScre
  GameOverScreen.GameOverScreen()
  Calls: None

5.5.9 HUD

The HUD class implements requirement 5(a), section 4.1.2 in the Requirements Document.

Fields:

1. List<Texture2D> powerUpIcons

No properties
Methods:

- HUD()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: The constructor for the HUD class. Instantiates a List of Texture2Ds, and loads the appropriate power-up graphics, storing them in the powerUpIcons list.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: GameRunningScreen.GameRunningScreen()
  Calls: None

- Draw() **Access modifier:** public
  Parameters: None
  Return value: None
  Description: Draws the HUD, drawing player lives and icons representing their available power-ups on the screen.
  Data it accesses: GameRunningScreen.GameObjects
  Preconditions: None
  Postconditions: Up-to-date player information has been drawn to the screen
  Called by: GameRunningScreen.Draw()
  Calls: None

5.5.10 Logic

The Logic class handles game rules that are not easily implemented in the CollisionCallback methods of the GameObjects, such as keeping track of whether all players have reached the finish point.

The Logic class implements requirement 6(a), section 4.1.2 in the Requirements Document.

Fields:

- private IScreen screen

Properties: None
Methods:

- Update()

  Access modifier: public
  Parameters: None

Return value: None
Description: This metod is responsible for updating game objects.
Data it accesses: Data in GameObjects.
Preconditions: None
Postconditions: The game has been updated according to the game rules.
Called by: Update() in the GameRunningScreen instance.
Calls: Update() on all GameObject instances.

5.5.11 Input

Fields:

- private ScaryMonsters game
- private List<Controller> controllers

Properties: None
Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: This metod is responsible for updating the Controllers.
  Data it accesses: None
  Preconditions: None
  Postconditions: The Controllers have been updated.
  Called by: Update() in an instance of any of the GameRunningScreen, CharacterSelectionScreen, GameStageSelectionScreen or GameOverScreen classes.
  Calls: Update() on the four Controller instances.

- PressedStart()

  Access modifier: public
  Parameters: None
  Return value: A Controller that pressed start, or null if no one pressed start.
  Description: Checks all four controllers for Start button presses, and returns one that did, or null if no one did.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: Logic.Update()
  Calls: None

5.5.12 Controller

The Controller class helps implement requirement 1(d), section 4.1.2 in the Requirements Document.

Fields:

- private Vector2 stick
- private bool aButton
- private bool leftTrigger
- private bool rightTrigger
- private bool startButton

Properties:

- public Vector2 Stick
- public bool Jump
- public bool Confirm
- public bool PowerupOne
- public bool PowerupTwo
- public bool Start

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the controller by checking the state of the corresponding gamepad through XNA's GamePad class, setting the fields of the controller as appropriate.
  Data it accesses: None
  Preconditions: None
  Postconditions: The controller has been updated.
  Called by: Input.Update()
  Calls: None

5.5.13 Physics

The Physics class implements requirement 2(c), section 4.1.2 in the Requirements Document.

Fields:

- private ScaryMonsters game

Properties:
Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the GameObjects positions, and velocities and checks for collisions, with the help of Farseer.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: Logic.Update()
  Calls: None

5.5.14 GameObjects

Fields:

- private ScaryMonsters game
- private List<GameObject> gameObjects
- private GameStage gameStage

Properties:

- public GameStage GameStage
- public List<GameObject> GameObjects
- public List<SpriteObject> SpriteObjects
- public List<ModelObject> ModelObjects

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates all GameObjects.
  Data it accesses:
  Preconditions: None
  Postconditions: All GameObjects are updated for this interation of the game loop.
  Called by: Logic.Update()
  Calls: Update() on all GameObjects in gameObjects.

5.5.15 Graphics

Fields:

- private ScaryMonsters game

Properties:
Methods:

- DrawGame()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the GameStage and all GameObjects as well as the head-up display.
  Data it accesses: None
  Preconditions: A race is in progress.
  Postconditions: The game has been drawn this iteration of the game loop.
  Called by: GameRunningScreen.Draw()
  Calls: DrawGameObjects(), DrawGameStage() and DrawHUD() on itself.

- DrawGameStage()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the platforms and background of the GameStage.
  Data it accesses: None
  Preconditions: The right settings on the GraphicsDevice has been set.
  Postconditions: The GameStage has been drawn this iteration of the game loop.

Called by: DrawGame()
Calls: None

- DrawHUD()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the head-up display.
  Data it accesses: None
  Preconditions: The right settings on the GraphicsDevice has been set.
  Postconditions: The head-up display has been drawn this iteration of the game loop.
  Called by: DrawGame()
  Calls: None

- DrawGameObjects()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the GameObjects.
  Data it accesses: None
  Preconditions: None
  Postconditions: The GameObjects has been drawn this iteration of the game loop.
  Called by: DrawGame(), DrawCharacterSelectScreen()
  Calls: DrawSpriteObjects(), DrawModelObjects()

- DrawSpriteObjects()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the SpriteObjects in the GameObjects list.
  Data it accesses: None
  Preconditions: The right settings on the GraphicsDevice has been set.
  Postconditions: The SpriteObjects has been drawn this iteration of the game loop.
  Called by: DrawGameObjects()
  Calls: None

- DrawModelObjects()

  Access modifier: public
  Parameters: None

Return value: None
Description: Draws the ModelObjects in the GameObjects list.
Data it accesses: None
Preconditions: The right settings on the GraphicsDevice has been set.
Postconditions: The ModelObjects has been drawn this iteration of the game loop.
Called by: DrawGameObjects()
Calls: None

- Draw(IScreen screen)

  Access modifier: public
  Parameters:
  IScreen screen - the screen that shall be drawn.
  Return value: None
  Description: Draws the specified screen.
  Data it accesses: None
  Preconditions: The right settings on the GraphicsDevice has been set.
  Postconditions: None
  Called by: IScreen.Draw()
  Calls: None

5.5.16 Camera

The Camera class implement requirement 5(b), 5(c) and 5(d), section 4.1.2 in the Requirements Document.

Fields:

- private Vector2 position

Properties:

- public Vector2 Position

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the camera position so that it follows the player in the lead.
  Data it accesses: None
  Preconditions: None

Postconditions: None
Called by: Logic.Update()
Calls: None

5.5.17 GameStage

A GameStage holds information about starting points, finish points and stationary platforms.

Fields:

- private Vector2[] startingPoints
- private int[][] platforms
- public FinishPoint finish
- public List<Monster> monsters
- public List<PowerupDispenser> powerupDispensers
- public List<MovingPlatform> movingPlatforms
- public List<SpringBoard> springBoards
- public List<Trap> traps

Properties:

- public Vector2[] StartingPoints
- public int[][] Platforms

No methods

5.5.18 GameObject

The GameObject class is an abstract class. It extends Geometry from the Farseer physics engine.

Fields:

- protected ScaryMonsters game

Properties: None

Methods:

- Update() (abstact method)

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the GameObject.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by:
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList) (abstract method)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: None
  Description: Takes care of collisions with other objects in an appropriate way.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by:
  Calls: None

5.5.19 ModelObject

The ModelObject class is an abstract class. It extends the GameObject class.

Fields:

- protected Model model

Properties:

- public Model Model

Methods: None

5.5.20 SpriteObject

The SpriteObject class is an abstract class. It extends the GameObject class.

No fields

- protected Texture2D sprite

Properties:

- public Texture2D Sprite

Methods: None

5.5.21 Effect

The Effect class is an abstract class that effects such as StunEffect or ShrinkEffect extends.

Fields:

- protected int timeLeft

No properties

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the timeLeft.
  Data it accesses: timeLeft
  Preconditions: None
  Postconditions: None
  Called by: Player.Update(), Monster.Update()
  Calls: None

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.
  Return value: None
  Description: Applies the effect to the specified game object.
  Data it accesses: None
  Preconditions: None

Postconditions: None
Called by: GameObject.Update()
Calls: None

5.5.22 StunEffect

The StunEffect class extends the Effect class. It helps implement requirements 2(g)iii and 3(g) of section 4.1.2 in the Requirements Document.

No fields

No properties

Methods:

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.
  Return value: None
  Description: Makes the target game object become stunned.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: GameObject.Update()
  Calls: None

5.5.23 ShrinkEffect

The ShrinkEffect class extends the Effect class. It helps implement requirement 4(d)vi.B of section 4.1.2 in the Requirements Document.

No fields

No properties

Methods:

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.
  Return value: None
  Description: Shrinks the target game object, slows its movment speed and lowers its jumping height.

Data it accesses: None
Preconditions: None
Postconditions: None
Called by: GameObject.Update()
Calls: None

5.5.24 SpeedEffect

The SpeedEffect class extends the Effect class. It helps implement requirement 4(d)iii.A of section 4.1.2 in the Requirements Document.

No fields

No properties

Methods:

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.
  Return value: None
  Description: Increase the speed of the target.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: GameObject.Update()
  Calls: None

5.5.25 LevitationEffect

The LevitationEffect class extends the Effect class. It helps implement requirement 4(d)iv.A of section 4.1.2 in the Requirements Document.

No fields

No properties

Methods:

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.
  Return value: None

Description: Reduce the gravitational forces on the target.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by: GameObject.Update()
Calls: None

5.5.26 InvulnerabilityEffect

The InvulnerabilityEffect class extends the Effect class. It helps implement requirement 3(f) of section 4.1.2 in the Requirements Document.

No fields

No properties

Methods:

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.
  Return value: None
  Description: Makes the target game object invulnerable.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: GameObject.Update()
  Calls: None

5.5.27 FetterEffect

The FetterEffect class extends the Effect class. It helps implement requirements 4(f)v.C and 4(f)v.E of section 4.1.2 in the Requirements Document.

No fields

No properties

Methods:

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.

Return value: None
Description: Slows the movement speed and reduces the jumping height of the target game object.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by: GameObject.Update()
Calls: None

5.5.28 ClawEffect

The ClawEffect class extends the Effect class. It helps implement requirement 4(d)ii.C of section 4.1.2 in the Requirements Document.

Fields:

1. private Player clawOwner

No properties

Methods:

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.
  Return value: None
  Description: Pulls the target game object towards the *clawOwner*. When the target game object has reached the *clawOwner* it sticks to the *clawOwner* and follows the her around.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: GameObject.Update()
  Calls: None

5.5.29 SlipEffect

The SlipEffect class extends the Effect class. It helps implement requirements 2(a)viii and 4(d)vii.C of section 4.1.2 in the Requirements Document.

No fields

No properties

Methods:

- Apply(GameObject target)

  Access modifier: public
  Parameters: GameObject target - the game object to which the effect shall apply itself.
  Return value: None
  Description: Decreases the rate of acceleration and deceleration of the target game object.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: GameObject.Update()
  Calls: None

5.5.30 Player

The Player class represents the player in the game, as specified in requirement 1(b), section 4.1.2 in the Requirements Document.

The Player class extends the ModelObject class.

Fields:

- private Enum Powerup None, Fetter, BoxingGlove, ...
- private Controller controller
- private Powerup powerup1
- private Powerup powerup2
- private List<Effect> effects
- private int lives
- private bool active

No properties

Methods:

- Update()
  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the player and takes approperiate action according to what buttons the player has pressed
  Data it accesses: effects, controller, powerup1, powerup2
  Preconditions: None

Postconditions: The player position and speed has been updated.
Called by:
Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
  Description: Takes care of collisions with other objects in an appropriate way.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by:
  Calls: None

- FirePowerup(int slot)

  Access modifier: private
  Parameters: slot - the slot which the power-up we want to fire is in
  Return value: None
  Description: Fires a powerup.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: Player.Update()
  Calls: None

- Run(int direction)

  Access modifier: Private
  Parameters: direction - the direction the player wants to run in
  Return value: None
  Description: Makes the player run.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: Player.Update()
  Calls: None

- Jump()

Access modifier: private
Parameters: None
Return value: None
Description: Makes the player jump.
Data it accesses: Geometry data field for speed
Preconditions: None
Postconditions: None
Called by: Player.Update()
Calls: None

- Respawn()

  Access modifier: private
  Parameters: None
  Return value: None
  Description: Takes care of everything that has do to with respawning (see Requirements Document, section 4.1.2, item 3(k)).
  Data it accesses: Geometry data field for position, active field
  Preconditions: None
  Postconditions: None
  Called by: Player.Update()
  Calls: None

5.5.31 Monster

The Monster class extends the ModelObject class.

The Monster class implement requirement 2(g), section 4.1.2 in the Requirements Document.

Fields:

- private List<Effect> effects
- bool stunnable

Properties: None

Methods:

- Monster(bool stunnable)

  Access modifier: public
  Parameters: bool stunnable - true if the monster can get stunned by a player when landed on
  Return value: None
  Description: Constructor for the Monster class.
  Data it accesses: It sets the stunnable field

Preconditions: None
Postconditions: None
Called by: Called once (for each monster) when the game stage is created. The game stage, along with monsters, is then stored on the hard drive and loaded each time a new race is going to be take place on the game stage.
Calls: None

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the monster and takes approperiate action
  Data it accesses: effects
  Preconditions: None
  Postconditions: The monster position and speed has been updated.
  Called by:
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
  Description: Takes care of collisions with other objects in an approperiate way.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by:
  Calls: None

- Run(int direction)

  Access modifier: Private
  Parameters: direction - the direction the monster wants to run
  Return value: None
  Description: Makes the monster run.
  Data it accesses: None
  Preconditions: None
  Postconditions: None

Called by: Monster.Update()
Calls: None

5.5.32 PowerupDispenser

The PowerupDispenser represents a power-up dispenser in the game. It extends the SpriteObject class.

The PowerupDispenser class implement requirement 2(d), section 4.1.2 in the Requirements Document.

Fields:

- private Enum Powerup Fetter, BoxingGlove, ...
- private List<Powerup> powerups
- private int current
- private int duration
- private int timeLeft

Properties: None

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the power-up dispenser, changing the current power-up after the specified time
  Data it accesses: current, timeLeft
  Preconditions: None
  Postconditions: The power-up dispenser has been updated
  Called by:
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and

let the objects pass through each other.
Description: Takes care of collisions with other objects in an approperiate way.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by:
Calls: None

5.5.33 Trap

The Trap class extends the ModelObject class.

The Trap class implement requirement 2(f), section 4.1.2 in the Requirements Document.

Fields:

- private bool active
- private int duration
- private int timeLeft

Properties: None

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the Trap, deciding whether it is going to be turned on or off.
  Data it accesses: active, timeLeft
  Preconditions: None
  Postconditions: The Trap has been updated.
  Called by: None
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided

Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
Description: Takes care of collisions with other objects in an appropriate way.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by:
Calls: None

5.5.34 MovingPlatform

The MovingPlatform class represents a platform moving cyclically along a path in the Game Stage described by a collection of XNA.FrameWork.Points.

The MovingPlatform class implement requirement 2(b)iv, section 4.1.2 in the Requirements Document.

Fields:

- private List<Point> path

Properties:

- public List<Point> Path

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the MovingPlatform, adjusting its movement direction to keep it moving towards the next node on the path.
  Data it accesses: None
  Preconditions: None
  Postconditions: The MovingPlatform has updated its movement direction
  Called by: None
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)

g2 - the object that this object collided with
contactList - a list of locations where they collided
Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
Description: Takes care of collisions with other objects in an appropriate way.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by:
Calls: None

5.5.35 Springboard

The Springboard class represents a springboard object in the game stage, as specified in the Requirements Document, section 4.1.2, item 2(b).

Springboard extends the ModelObject class.

No fields

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the animation of the springboard.
  Data it accesses: None
  Preconditions: None
  Postconditions: The animation of the springboard has been updated.
  Called by: Logic.Update()
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.

Description: Takes care of collisions with other objects in an approperiate way. If the collision is with a player or monster, then it sets the speed of the player/monster upwards to the speed specified in the Requirements Document, section 6, item 1(b)i.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by: PhysicsSimulator collision event
Calls: None

5.5.36 FinishPoint

The FinishPoint class represents a finishing point in a game stage, as specified in the Requirements Document, section 4.1.2, item 2(a)iv.

FinishPoint extends the SpriteObject class.

No fields

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the animation of the FinishPoint
  Data it accesses: None
  Preconditions: None
  Postconditions: The animation of the FinishPoint has been updated
  Called by: Logic.Update()
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
  Description: Takes care of collisions with other objects in an approperiate way. If the collision is with a player then play a special animation and play a sound.
  Data it accesses: None

Preconditions: None
Postconditions: The animation and sounds effects are played.
Called by: PhysicsSimulator collision event
Calls: None

5.5.37 BananaPeel

The BananaPeel class represents a banana peel from the Banana Peels power-up, as specified in the Requirements Document, section 4.1.2, item 4(d)vii.

BananaPeel extends the SpriteObject class.

Fields:

- private Player owner

No properties

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the animation of the object
  Data it accesses: None
  Preconditions: None
  Postconditions: The animation has been updated
  Called by: Logic.Update()
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
  Description: Takes care of collisions with other objects in an apropeiate way. If the collision is with a player/monster that player will have it's acceleration/deceleration reduced and that player/monster will be stunned (6.4(g)iii).
  Data it accesses: None

Preconditions: None
Postconditions: None
Called by: PhysicsSimulator collision event
Calls: None

5.5.38 Fetter

The Fetter class represents a fetter from the Fetter power-up, as specified in the Requirements Document, section 4.1.2, item 4(d)v.

Fetter extends the ModelObject class.

Fields:

- private int placedBy

No properties

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the animation of the object
  Data it accesses: None
  Preconditions: None
  Postconditions: The animation has been updated
  Called by: Logic.Update()
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
  Description: Takes care of collisions with other objects in an appropriate way. If the collision is with a player/monster the fetter will attach itself to it and slow it down (6.4(e)ii).
  Data it accesses: None
  Preconditions: None
  Postconditions: None

Called by: PhysicsSimulator collision event
Calls: None

5.5.39 BoxingGlove

The BoxingGlove class represents a boxing glove from the Boxing Glove power-up, as specified in the Requirements Document, section 4.1.2, item 4(d)i.

BoxingGlove extends the ModelObject class.

No fields

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the position of the glove
  Data it accesses: None
  Preconditions: None
  Postconditions: The position has been updated
  Called by: Logic.Update()
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
  Description: Takes care of collisions with other objects in an appropriate way. If the collision is with a player/monster, it will get a vertical speed in the same direction the glove was going. (6.4(a)ii).
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: PhysicsSimulator collision event
  Calls: None

5.5.40 ShrinkingRay

The ShrinkingRay class represents a shrinking ray from the Shrinking Ray power-up, as specified in the Requirements Document, section 4.1.2, item 4(d)vi.

ShrinkingRay extends the SpriteObject class.

No fields

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the position of the shrinking ray
  Data it accesses: None
  Preconditions: None
  Postconditions: The position has been updated
  Called by: Logic.Update()
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
  Description: Takes care of collisions with other objects in an approperiate way. If the collision is with a player/monster, it will become shrunk and have its maximum speed and jumping height reduced (4(d)vi.(b)).
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: PhysicsSimulator collision event
  Calls: None

5.5.41 Claw

The Claw class represents a claw from the Claw power-up, as specified in the Requirements Document, section 4.1.2, item 4(d)ii.

Claw extends the SpriteObject class.

No fields

Methods:

- Update() **Access modifier:** public
  Parameters: None
  Return value: None
  Description: Updates the position of the claw
  Data it accesses: None
  Preconditions: None
  Postconditions: The position has been updated
  Called by: Logic.Update()
  Calls: None

- CollisionCallback(Geometry g1, Geometry g2, ContactList contactList)

  Access modifier: public
  Parameters:
  g1 - this object (needed because of the format of the delegate methods that can register to the collision event)
  g2 - the object that this object collided with
  contactList - a list of locations where they collided
  Return value: bool - false if Farseer should ignore the collision and let the objects pass through each other.
  Description: Takes care of collisions with other objects in an appropriate way. If the collision is with a player/monster the claw will grab it to itself (6.4(b)ii(b)) and the player becomes stunned (6.4(b)ii(a))).
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by: PhysicsSimulator collision event
  Calls: None

5.5.42 Path

The Path class represents a path that an object, such as a moving platform (see Requirements Document, section 4.1.2, item 2(b)) or moving traps (see Requirements Document, section 4.1.2, item 2(f)) can follow. It includes nodes (positions) and speeds (the object following the path can change its speed between nodes).

Fields:

- private List<Vector2> points
- private List<double> speeds

Properties:

- private List<Vector2> points
- private List<double> speeds

No methods

5.6 Table of References

The following is a table showing where the functional requirements (found in the Requirements Document) are being implemented. If a functional requirement is listed more than once, it is because it is being implemented by more than one class or method.

Requirement	**Class**	**Method**
2(a)viii	GameStage	-
2(a)viii	Player	Update()
2(a)viii	SlipEffect	Apply()
2(b)i	Monster	Update()
2(b)i	Player	Update()
2(b)ii	GameStage	-
2(b)iii	GameStage	-
2(b)iv	MovingPlatform	-
2(b)iv	Path	-
2(b)iv	GameStage	-
2(b)v	GameStage	-
2(c)	Physics	Update()
2(d)i	PowerupDispenser	Update()
2(d)ii	PowerupDispenser	CollisionCallback()
2(d)iii	PowerupDispenser	CollisionCallback()
2(e)	GameStage	-
2(f)i	Player	CollisionCallback
2(f)ii	Trap	Update()
2(f)iii	Trap	Update()
2(f)iii	Path	-
2(g)i	Monster	Update()
2(g)ii	Physics	Update()
2(g)iii	StunEffect	Apply()
2(g)iii	Monster	-
2(g)iv.A	Physics	Update()
2(g)iv.B	Player	CollisionCallback()
2(g)iv.C	Player	CollisionCallback()
2(g)iv.C	Monster	CollisionCallback()
2(g)iv.C	StunEffect	Apply()
2(g)iv.D	Player	CollisionCallback()
2(g)v	Monster	CollisionCallback()
3(a)i	Player	Update()
3(a)ii	Player	CollisionCallback()
3(a)iii	Player	Update()
3(a)iv	Player	Update()
3(a)v	Physics	Update()

Requirement	**Class**	**Method**
3(b)	Player	Run()
3(c)	Player	Jump()
3(d)	Player	CollisionCallback()
3(e)	Player	FirePowerup()
3(f)	InvulnerabilityEffect	Apply()
3(f)	Player	Update()
3(g)	StunEffect	Apply()
3(g)	Player	Update()
3(h)	Player	CollisionCallback()
3(i)	Player	CollisionCallback()
3(j)	Player	Update()
3(k)	Player	Respawn()
4(a)	PowerupDispenser	-
4(b)	Player	FirePowerup()
4(c)	Player	FirePowerup()
4(d)i	BoxingGlove	-
4(d)i.A	BoxingGlove	Update()
4(d)i.B	BoxingGlove	CollisionCallback()
4(d)i.B	Player	CollisionCallback()
4(d)i.C	BoxingGlove	CollisionCallback()
4(d)i.C	BoxingGlove	Update()
4(d)ii	Claw	-
4(d)ii.A	Claw	Update()
4(d)ii.B	Claw	CollisionCallback()
4(d)ii.B	Claw	Update()
4(d)ii.C	ClawEffect	Apply()
4(d)ii.C	Player	CollisionCallback()
4(d)ii.D	Player	Update()
4(d)ii.E	Player	Update()
4(d)ii.F	ClawEffect	-
4(d)iii.A	Player	FirePowerup()
4(d)iii.A	SpeedEffect	Apply()
4(d)iv.A	Player	FirePowerup()
4(d)iv.A	LevitationEffect	Apply()
4(d)v.A	Player	FirePowerup()
4(d)v.B	Physics	Update()
4(d)v.C	Fetter	Update()
4(d)v.D	Fetter	Update()
4(d)v.E	FetterEffect	Apply()

Requirement	**Class**	**Method**
4(d)vi.A	ShrinkingRay	Update()
4(d)vi.B	ShrinkingRay	CollisionCallback()
4(d)vi.B	ShrinkEffect	Apply()
4(d)vi.C	ShrinkingRay	CollisionCallback()
4(d)vi.C	ShrinkingRay	Update()
4(d)vii.A	Player	FirePowerup()
4(d)vii.B	Physics	Update()
4(d)vii.C	BananaPeel	CollisionCallback()
4(d)vii.C	StunEffect	Apply()
4(d)vii.C	SlipEffect	Apply()
4(d)vii.D	BananaPeel	CollisionCallback()
5(a)	HUD	-
5(b)	Camera	-
5(c)	Camera	Update()
5(d)	Camera	Update()
5(e)	Player	Update()
6(a)	Logic	Update()
6(b)	Logic	Update()
6(b)	Player	-
6(c)	Logic	Update()
8(a)	CharacterSelectionScreen	-
8(d)	GameStageSelectionScreen	-
8(e)	GameStageSelectionScreen	-
9(a)	GameRunningScreen	Update()
9(b)	GameRunningScreen	Update()
9(c)	GameRunningScreen	-
10(a)	GameOverScreen	-

6 Functional Test Cases

Since almost all of the functional requirements in our software have to do with graphical representations of the game state and user interaction we will not use regular automated assertion tests. Rather, all tests will need to be performed by a tester who monitors and interacts with the game in order to assert that the test is successful. Our source code will include a preprocessor debug switch—activating the switch and compiling and running the game will result in starting a special debug version of the game.

The debug version of the game has all the features of the regular game, but with a few changes an additional features:

- The regular game stages are not available in the debug version. Instead, a selection of game stages constructed especially for debugging purposes are made available to the tester.
- The debug version skips all menus, instead starting directly on a default debug game stage. The tester can change between the available debug game stages by pressing buttons on the keyboard or the Xbox 360 gamepad.
- By default two players are instantiated, both of which can be controlled by the tester.
- The debug version is designed to be able to run on both the PC and the Xbox 360 console. The game shall run in windowed mode on the PC.
- The debug version can receive input from the keyboard and Xbox 360 gamepads. Several players can be controlled by the same keyboard, but two Xbox 360 controllers are needed in order to control both players.
- A player can be given additional lives at any time by pushing a button on the keyboard or the Xbox 360 gamepad
- A player can switch the contents of her first power-up slot between all available power-ups by pushing a button on the keyboard or the Xbox 360 gamepad, giving the tester unlimited access to all power-ups
- The debug game stage can be reinitialized at any time by pushing a button on the keyboard or the Xbox 360 gamepad
- The debug version displays the average time it took to render the last couple of frames. The amount of frames to be averaged is defined as a constant.

The test cases are distributed across several different debug game stages, designed to make testing convenient.

6.1 Any Test Game Stage

The following test cases can be performed on any test game stage.

- **Functionality being tested**: A player is always able to change her horizontal direction except while stunned.
  Functional requirement: 3(a)iv.
  Expected output: The horizontal direction changes, except when stunned.
  Procedure: While running/walking/standing still/falling/jumping try to change the horizontal direction of the player, verify that it is possible. Get stunned (for example, let the second player jump fire a boxing glove at the first player) and try to change the horizontal direction of the player and verify that it is not possible.

- **Functionality being tested**: When a player does not receive any horizontal input the player's horizontal speed decreases until she has come to a standstill.
  Functional requirement: 3(a)iii.
  Expected output: The horizontal speed is decreased to 0 when no horizontal input is received from the player.
  Procedure: While in a horizontal movement (for example running) let go of the game pad and verify that the player comes to a standstill.

- **Functionality being tested**: Two players can not occupy the same space
  Functional requirement: 3(a)ii.
  Expected output: The player trying to occupy the other players space will be pushed back in the same way as if she was colliding with a wall.
  Procedure: Verify that when running/walking/jumping into another player, the two players do not overlap.

- **Functionality being tested**: Vertical and horizontal movement shall be independent
  Functional requirement: 3(a)i.
  Expected output: The horizontal speed of a player is not affected by jumping or falling off a platform.
  Procedure: While running, jump or fall of a platform and verify that the player maintains her horizontal movement speed.

- **Functionality being tested**: That platforms are unaffected by gravity.
  Functional requirement: 2(b)ii.
  Expected output: The platforms remain suspended in the air.
  Procedure: Verify that the platforms remain suspended in the air.

- **Functionality being tested**: Player being affected by gravity.
  Functional requirement: 3(a)v.
  Expected output: Players that are airborne will fall downwards.
  Procedure: Walk off of the edge of a platform and verify that the player starts falling downwards. Jump and verify that the player eventually decelerates and starts falling.

- **Functionality being tested**: Maximum running speed.
  Functional requirement: 3(b)i.
  Expected output: Running players will stop accelerating when they have reached their maximum running speed.
  Procedure: Start running in any direction and verify that the player eventually ceases to accelerate.

- **Functionality being tested**: Players accelerate and decelerate.
  Functional requirement: 3(b)ii.
  Expected output: When standing still and starting to run the speed shall increase constantly until the maximum speed is reached. When a player changes horizontal direction she should first decelerate to a standstill and then start accelerating in the new direction.
  Procedure: Start running in any direction and verify that the player's horizontal speed increases linearly. While running, change direction and verify that the player decelerates to a standstill and then starts accelerating in the new direction.

- **Functionality being tested**: Player must be on a platform to be able to jump.
  Functional requirement: 3(c)i.
  Expected output: Players are only able to jump when standing on platforms.
  Procedure: Try to jump while airborne and verify that the player does not gain any vertical speed.

- **Functionality being tested**: Players shall be able to jump.
  Functional requirement: 3(c)ii.
  Expected output: When a player jumps she instantly get an upwards velocity, which diminishes linearly and eventually makes the player fall downwards.
  Procedure: While moving or standing still push the jump button on the game pad and verify that it instantly affects the vertical upwards speed of the player. Also verify that the players vertical speed decreases, eventually making her fall downwards.

- **Functionality being tested**: Player shall be able to use a power-up from the inventory at any moment, except while stunned.
  Functional requirement: 3(e).

Expected output: The player uses the power-up in case she is not stunned.
Procedure: Try using a power-up and verify that the power-up is activated in all situations except when stunned.

- **Functionality being tested**: While stunned a player is unable to affect her own movement.
  Functional requirement: 3(g)i and 3(g)iii.
  Expected output: When stunned all player inputs are ignored.
  Procedure: Get stunned (for example, let the second player jump fire a boxing glove at the first player) and verify that the player does not respond to input.

- **Functionality being tested**: While stunned a player starts decelerating.
  Functional requirement: 3(g)ii.
  Expected output: Stunned players decelerate.
  Procedure: Start running with the first player and have the second player fire a boxing glove that will hit the first player. Verify that the first player starts decelerating after being hit by the boxing glove.

- **Functionality being tested**: Players should be able to stun each other by jumping on each others head.
  Functional requirement: 3(h).
  Expected output: When a player jumps upon another player's head the target becomes stunned.
  Procedure: Jump on a player's head and verify that she becomes stunned.

- **Functionality being tested**: If a player ends up outside the screen, she shall lose a life and then respawn. **Functional requirement**: 3(j).
  Expected output: The player will respawn and lose a life when ending up outside the screen.
  Procedure: Move one player towards the right end of the screen and verify that the camera starts following her. Continue doing so until the other player ends up outside the screen. Verify that this causes the player that ended up outside the screen to lose a life. Verify that the player that ended up outside the screen respawns.

- **Functionality being tested**: That the fetter power-up is placed close to the player.
  Functional requirement: 4(d)v.A and 4(d)v.B.
  Expected output: The fetter is spawned at the player's position and is affected by gravity.

Procedure: Use a Fetter power-up and verify that the fetter is spawned at the position of the player.

- **Functionality being tested**: That the speed boost power-up makes the player run faster.
  Functional requirement: 4(d)iii.
  Expected output: The player can run faster, but the time to accelerate to maximum speed, and the time it takes to stop is the same as before.
  Procedure: Have the first player use a Speed Boost power-up and start running with both players. Verify that the first player moves faster than the second player.

- **Functionality being tested**: Fetter is affected by gravity.
  Functional requirement: 4(d)v.B.
  Expected output: When dropping a fetter from the air it shall fall.
  Procedure: Use a Fetter power-up while in mid-air. Verify that the fetter is spawned at the position of the player and that it falls to the ground

- **Functionality being tested**: The fetter shall not catch the player who activated it.
  Functional requirement: 4(d)v.D.
  Expected output: When colliding with a fetter you yourself have activated nothing should happen.
  Procedure: Activate a Fetter and collide with it. Verify that nothing changes.

- **Functionality being tested**: That the amount of lives a player has is presented on the screen.
  Functional requirement: 5(a).
  Expected output: The right amount of lives a player has is presented on the screen.
  Procedure: Collide with a monster and verify that you now have one life less than before. Give the player an extra life and verify that you now have one more life.

- **Functionality being tested**: That a players power-ups are shown on the screen.
  Functional requirement: 5(a), 4(a), 4(b) and 4(c).
  Expected output: The right power-ups are shown on the screen.
  Procedure: Take two power-ups from the power-up dispenser and verify that they are both shown. Use both power-ups. Do this procedure for all power-ups in the game, and verify that the icons for all the power-ups are right.

- **Functionality being tested**: That the camera follows the player in the lead, but gives a buffer of 1/3 of screen.
  Functional requirement: 5(d)
  Expected output: The camera moves only if the leader moves, and it maintains a distance of one third of a screen between the player in the lead and the rightmost edge of the screen.
  Procedure: Spawn another player, and run with one of the players to the right. Verify that the camera follows the player in the lead, and that it keeps a buffer of 1/3 of the screen to the right.

- **Functionality being tested**: That a player dies if she gets outside the screen.
  Functional requirement: 5(e)
  Expected output: The player dies when she is outside the screen.
  Procedure: Have the first player run to the right until the second player ends up outside the screen. Verify that the second player loses a life and respawns.

- **Functionality being tested**: That players start with 3 lives.
  Functional requirement: 6(b) and system requirement 6(a).
  Expected output: Each player has three lives at the start of the game.
  Procedure: Verify that each player starts with 3 lives by looking at how many lives are displayed on the screen and checking if the game ends after both players have died three times.

- **Functionality being tested**: That a player dies if he doesn't have any lives left.
  Functional requirement: 6(b) and system requirement 6(a).
  Expected output: The player dies after losing all her lives.
  Procedure: Walk into a monster 3 times and validate that the player has died, and forfeited the race.

- **Functionality being tested**: If a player is the only one left alive she wins the race.
  Functional requirement: 6(c)
  Expected output: The only player left wins.
  Procedure: Make the second player lose all of her lives and verify that this causes the first player to win the race.

6.2 Debug Game Stage 1

This game stage features a monster and a moving platform. The game stage is constructed in a manner so that the monster's path is blocked by short walls on its left and right hand side. There is also a moving platform on the game stage that will occasionally intercept the monster in its path.

- **Functionality being tested**: The Shrinking Ray
  Functional requirement: 4(d)vi
  Expected output: The shrinking ray travels horizontally, disappears when it hits a player/monster/wall and shrinks players/monsters. Players that are shrunk have a slower maximum speed, and shorter jumping height.
  Procedure: Shoot a Shrinking Ray at a wall and verify that it disappears. Shoot a Shrinking Ray at a monster and verify that its behaviour changes as expected. Shoot the second player with the Shrinking Ray, walk/jump around with the other player and verify that the player's behaviour is changed as expected.

- **Functionality being tested**: The effect of the Fetter power-up.
  Functional requirement: 4(d)v.A and 4(d)v.E.
  Expected output: A player/monster with a fetter attached has a slower maximum running speed than normal and can not jump as high as usual.
  Procedure: Have the first player use a Fetter power-up and make the second player collide with the Fetter. Verify that the Fetter becomes attached to the second player. Jump with the second player and verify that the height of her jump is less than without the fetter. Run with the second player and verify that her maximum running speed is slower than usual. Make a monster collide with a Fetter and verify that the speed of the monster is less when the Fetter is attached.

- **Functionality being tested**: That the Boxing Glove power-up travels horizontally in the direction the player is facing, that the player/monster it hits gets a horizontal speed in the same direction and that the glove disappears when it hits an object.
  Functional requirement: 4(d)i.
  Expected output: The Boxing Glove travels horizontally in the direction the player is facing. When it hits a monster/ player it gives them a horizontal speed in the traveling direction and then disappears. If it hits a wall it shall only disappear.
  Procedure: Spawn a player with a Boxing Glove power-up and shoot it at a monster/player/wall and very that the expected output happens.

- **Functionality being tested**: If a player lands on another player or a monster she shall automatically jump.
  Functional requirement: 3(i).
  Expected output: When landing on a opponent or monster the player will automatically jump.
  Procedure: Jump on a player or a monster that gets stunned when

jumped on, verify that the player jumps directly after landing on the monster or player.

- **Functionality being tested**: Graphical representation of the stun effect.
  Functional requirement: 3(f)iii and 3(g)iv.
  Expected output: Invulnerability and stunned status are graphically represented.
  Procedure: Run into a monster. Verify that is is visible that the player is stunned and invulnerable.

- **Functionality being tested**: Using power-ups on monsters.
  Functional requirement: 2(f)v.
  Expected output: Power-ups affect the monsters in the same way they would a player.
  Procedure: Using the same procedure as when testing power-ups on a player, test all power-ups on a monster and confirm that they have the same effect as they would have on a player.

- **Functionality being tested**: Losing life when running into monsters.
  Functional requirement: 2(f)iv.B.
  Expected output: The player loses a life when touching the monster.
  Procedure: Run into a monster and confirm that a life was subtracted from the player's life total.

- **Functionality being tested**: That players or monsters can't pass through platforms in any way.
  Functional requirement: 2(b)i
  Expected output: The players and monster(s) don't pass through the platforms in any way.
  Procedure: Verify that the player or monster(s) do not fall through the platforms. Try to run through a wall and verify that the player doesn't pass through it. Verify that the monster does not move through the platforms. Shoot the other player or the monster with the Boxing Glove power-up and verify that the player or monster does not pass through platforms when hit by the Boxing Glove power-up.

- **Functionality being tested**: That moving platforms move along a path correctly.
  Functional requirement: 2(b)iv
  Expected output: The moving platform moves along its path.
  Procedure: Verify that the platform moves along its path.

- **Functionality being tested**: That players or monsters can not pass through moving platforms in any way.
  Functional requirement: 2(b)i
  Expected output: The player ad monster do not pass through the moving platform in any way.
  Procedure: Jump onto the moving platform. Verify that the player does not fall through the moving platform. Try to run through a wall on the moving platform. Verify that the player cannot run through the moving platform.

- **Functionality being tested**: Players shall be able to stun monsters.
  Functional requirement: 2(f)iii.
  Expected output: The monster gets stunned. The monster stops for the duration of the stun and the player does not lose life as a result of touching the stuenned monster.
  Procedure: Fire a Boxing Glove at the monster and confirm that it stops moving and that the visual stun effect is displayed. Touch the monster and confirm that this does not cause the player to lose life.

6.3 Debug Game Stage 2

A game stage with several monsters next to each other. The game stage is constructed in a manner so that the monsters' paths are blocked by walls on their left and side and by a bottomless pit at their right. Some of the monsters will initially start by moving to the right while the others will be moving to the left, which means that the monsters will eventually start colliding with each other.

- **Functionality being tested**: Banana Peels stunning power.
  Functional requirement: 4(d)vii.C.
  Expected output: When a player or a monster trips on a Banana Peel they will become stunned and have a lower acceleration and deceleration than usual.
  Procedure: Equip one player with a Banana Peel power-up. Activate the power-up and make the player and the monster collide with the banana peels. Verify that the player and the monster become stunned and that their acceleration / deceleration is affected in the manner intended.

- **Functionality being tested**: Banana peels
  Functional requirement: 4(d)vii.A, 4(d)vii.B and 4(d)vii.D.
  Expected output: When activating the Banana Peel power-up 3 small Banana Peels will appear near the player who activated it. If the player is in the air the peels will fall down due to gravity. When the

player who activated the power-up steps on the peels nothing happens.
Procedure: Spawn a player with the Banana Peel power-up. Jump with the player and activate the power-up while in the air. Verify that three banana peels appear and are positioned near the player. Verify that the peels fall down to the ground. Also verify that when stepping on the peels nothing changes.

- **Functionality being tested**: A player shall respawn near the middle of the screen.
  Functional requirement: 3(k).
  Expected output: When a player falls down a bottomless pit or ends up outside the screen she shall respawn somewhere in the middle of the screen.
  Procedure: Fall down a bottomless pit and verify that the player respawns near the middle of the screen. End up outside the screen and verify that the player respawns near the middle of the screen.

- **Functionality being tested**: Monsters move back and forth on platforms.
  Functional requirement: 2(g)i.A.
  Expected output: The monster moves back and forth, turning around when reaching an obstruction or the end of the platform.
  Procedure: Observe the monster and confirm that it changes direction when reaching an obstruction or the end of the platform.

- **Functionality being tested**: Players should not be able to pass through monsters and monsters should not be able to pass through other monsters.
  Functional requirement: 2(f)iv.A.
  Expected output: Players and monsters do not pass through each other.
  Procedure: Observe two monsters moving towards each other and confirm that they change direction when they reach each other. Try running through a monster and confirm that this is not possible. Shoot a Boxing Glove at a monster and confirm that the monster does not move through the other monster or a player when moved by the Boxing Glove.

- **Functionality being tested**: Jumping on monsters.
  Functional requirement: 2(f)iv.C.
  Expected output: The player loses a life when landing on the monster or the monster gets stunned, depending on the type of monster.
  Procedure: Jump on a monster. If it is a type of monster that

the player can jump on without losing life, confirm that the player did not lose life and that the monster was stunned. If the monster was a type of monster that causes players to lose life when jumping on them, confirm that a life was subtracted from the player's life total.

6.4 Debug Game Stage 3

A game stage that starts with a monster suspended in the air above the first players' starting points. (When the game starts the monster should begin to fall.)

- **Functionality being tested**: Gravitation in the game
  Functional requirement: 2(c)
  Expected output: Players and monsters fall towards the bottom of the screen when in midair.
  Procedure: Let the player walk off the platform and observe if the player starts falling. Observe wheter the monster falls down or not.

- **Functionality being tested**: Losing life from monsters located above the player.
  Functional requirement: 2(f)ivD.
  Expected output: The player loses a life when touching the monster.
  Procedure: Let a monster higher up than the player touch the player and confirm that a life was subtracted from the player's life total.

6.5 Debug Game Stage 4

A game stage with a power-up dispenser placed close to the players' starting points.

- **Functionality being tested**: The power-up dispenser shall cycle through available power-ups
  Functional requirement: 2(d)i.
  Expected output: The power-up dispenser cycles through all available power-ups, both in respect to the icons displayed and the power-up the player gets when touching the dispenser.
  Procedure: Observe the power-up dispenser and check if it cycles through all the power-up icons.

- **Functionality being tested**: The player shall gain the power-up that is currently shown on the power-up dispenser

Functional requirement: 2(d)ii.
Expected output: The player gets the power-up that is shown on the power-up dispenser when she touches the power-up dispenser
Procedure: For each power-up displayed on the power-up dispenser, touch the power-up dispenser and confirm that you got the power-up corresponding to the power-up icon that was currently shown.

- **Functionality being tested**: The player shall only be able to use the same power-up dispenser once
  Functional requirement: 2(d)iii.
  Expected output: The player gets a power-up from the power-up dispenser the first time she touches it and no power-ups from touching it subsequent times.
  Procedure: Touch the power-up dispenser several times and confirm that the player got only one power-up.

A game stage with a power-up dispenser placed close to the players' starting points.

- **Functionality being tested**: The power-up dispenser shall cycle through available power-ups
  Functional requirement: 2(d)i.
  Expected output: The power-up dispenser cycles through all available power-ups, both in respect to the icons displayed and the power-up the player gets when touching the dispenser.
  Procedure: Observe the power-up dispenser and check if it cycles through all the power-up icons.

- **Functionality being tested**: The player shall gain the power-up that is currently shown on the power-up dispenser
  Functional requirement: 2(d)ii.
  Expected output: The player gets the power-up that is shown on the power-up dispenser when she touches the power-up dispenser
  Procedure: For each power-up displayed on the power-up dispenser, touch the power-up dispenser and confirm that you got the power-up corresponding to the power-up icon that was currently shown.

- **Functionality being tested**: The player shall only be able to use the same power-up dispenser once
  Functional requirement: 2(d)iii.
  Expected output: The player gets a power-up from the power-up dispenser the first time she touches it and no power-ups from touching it subsequent times.
  Procedure: Touch the power-up dispenser several times and confirm that the player got only one power-up.

6.6 Debug Game Stage 5

A game stage containing a static trap and a moving trap, and a monster.

- **Functionality being tested**: Whenever a player loses a life it shall become invulnerable for a short time, in that time she may not lose any lives.
  Functional requirement: 3(f)i and 3(f)ii.
  Expected output: When a player is invulnerable nothing except ending up outside the screen may cause her to lose a life.
  Procedure: Become invulnerable (by colliding with a monster, for example) and verify that colliding with monsters

- **Functionality being tested**: Traps shall cause the player to lose a life when touching them.
  Functional requirement: 2(f)i.
  Expected output: The player loses a life when touching the trap.
  Procedure: Touch a trap and confirm that a life was subtracted from the player's life total.

- **Functionality being tested**: Traps shall be able to turn themselves on and off.
  Functional requirement: 2(f)ii.
  Expected output: The player does not lose life when touching a trap in its off state.
  Procedure: Touch a trap in its off state and verify that no life was subtracted from the player's life total.

- **Functionality being tested**: Traps shall either move along a set path or have a fixed location
  Functional requirement: 2(f)iii.
  Expected output: Traps that move move along their paths correctly, traps that are not supposed to move stay stationary
  Procedure: Observe moving and non-moving traps and confirm that they move in the manner intended.

6.7 Debug Game Stage 6

A game stage with a finish point close to the players' starting point.

- **Functionality being tested**: That the game ends if all players have finished or died.
  Functional requirement: 6(a)

Expected output: The game ends.
Procedure: Go into the finishing point with both players. Verify that the game ends. Restart the game stage. Go into the finish point with one player. Die with the other (lose all of the player's lives). Verify that the game ends.

6.8 Regular Game Stages

The following test cases can be performed on any regular game stage.

- **Functionality being tested**: That two players cannot select the same character
  Functional requirement: 8(a)
  Expected output: After one player has chosen a character, another player cannot choose the same character.
  Procedure: Start the game with two gamepads. Choose the first character with the first gamepad and verify that the same character cannot be chosen with the other gamepad.

- **Functionality being tested**: There are at least 3 game stages to choose from.
  Functional requirement: 8(b)
  Expected output: In the game stage selecting screen, you are able to select at least 3 different game stages.
  Procedure: Start the game with two gamepads, choose characters for both players and on the next screen, verify that there are at least 3 different game stages available.

- **Functionality being tested**: Any player can pause the game.
  Functional requirement: 9(a)
  Expected output: Both players can pause the game.
  Procedure: Start a race with two players. Verify that both players can pause the game.

- **Functionality being tested**: Only the player who has paused the game can unpause it.
  Functional requirement: 9(b)
  Expected output: Only the player who has paused the game can unpause it.
  Procedure: Start a race with two players. Pause the game with one gamepad. Verify that the other gamepad cannot be used to unpause the game.

- **Functionality being tested**: That the pause-screen contains "Resume", "Restart" and "Abort".
  Functional requirement: 9(c)

Expected output: The pause-screen contains “Resume”, “Restart” and “Abort”.
Procedure: Start a race with two players. Pause the game and verify that “Resume”, “Restart” and “Abort” are all options.

- **Functionality being tested**: That there is a screen which shows finish times and places after the race has finished.
  Functional requirement: 10(a)
  Expected output: There is a screen which shows finish times and places after the race has finished.
  Procedure: Finish a race and verify that there is a screen showing finish times and places.

- **Functionality being tested**: That there are sound effects in the game.
  Functional requirement: 7(b)
  Expected output: There are sound effects for jumping, losing a life, reaching the goal, using a power-up, landing on another player, landing on a platform, landing on a monster.
  Procedure: Verify that there is a sound effect after these procedures:
  - Jump with the player
  - Walk into a monster
  - Take a power-up from a power-up dispenser, and use it.
  - Jump and land on a platform
  - Jump and land on a monster
  - Spawn another player and jump on him
  - Jump into the finishing portal

- **Functionality being tested**: Players shall be able to avoid monsters either by jumping over them, waiting or going around them.
  Functional requirement: 2(f)vi.
  Expected output: A player can go through the game stage without losing any lives from touching monsters.
  Procedure: Play each game stage and verify that it would be possible to reach the finish point without losing any lives by colliding with a monster.

- **Functionality being tested**: Players shall be able to avoid traps either by jumping over them, waiting them out or going around them.
  Functional requirement: 2(f)iv.
  Expected output: A player can go through the game stage without losing any lives from touching traps.

Procedure: Play each game stage and verify that it would be possible to reach the finish point without losing any lives by means of traps.