Multiplayer Platform Game
Group 19

Martin Petterson
Oskar Kvist
Christoffer Ekeroth
Misael Berrios Salas

February 18, 2008

Contents

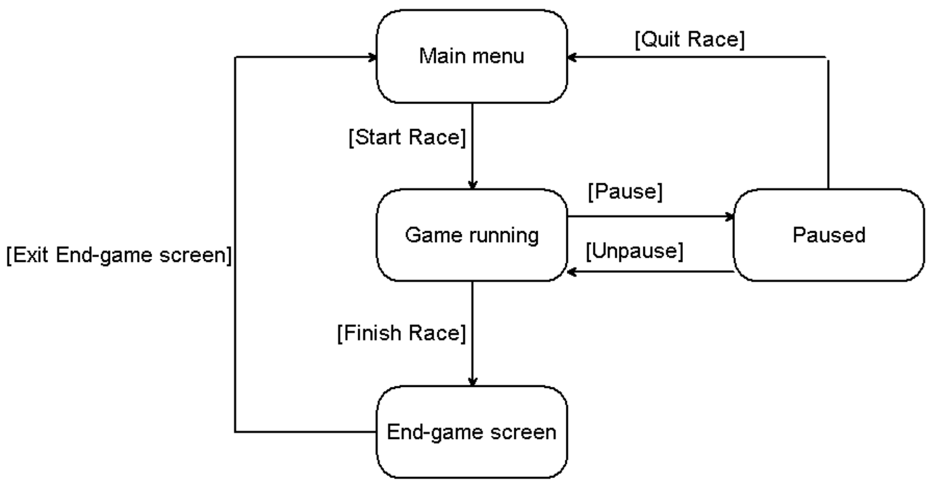
1 Introduction

2 System Overview

2.1 General Description

2.2 Overall Architecture Description

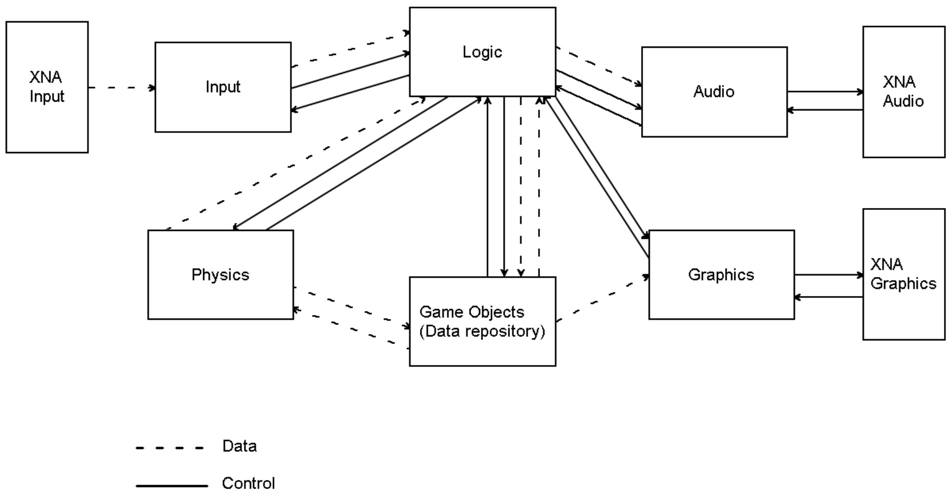
The following is a state transition diagram showing the states the game can be in as well as how the game can change state.



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

- *Main Menu*—This is the initial menu presented to the players upon startup and before the start of each race. In order to start a race players select characters and a game stage.
- *Game Running*—In this state the players play the actual game (sometimes referred to as a "race").
- *Paused*—In this state the race is paused and the players are presented with a menu.
- *End-Game Screen*—In this state the players are presented with the positions they finished in and their times.

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* is responsible for enforcing game rules on the world objects as well as handling menu navigation.

  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 Logic module gets button states from the Input module and events from the Physics module (collisions between objects, etc.) and writes to the Game Objects module, changing the state of and making new game objects. The Logic module also calls the Audio module, telling it when to play sound effects, and the Graphics module, telling it to update the screen.

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

- The *Audio module* is responsible for playback of music and sound effects. The Audio module calls the XNA Audio module to play music and sound effects.

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

  The Physics module differs from the Logic module in that the Physics module applies rules of physics upon the game objects while the Logic module applies other game rules upon the game.

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 Game Objects repository and utilizes calls to the XNA Graphics module to draw the appropriate representations of the Game Objects to the screen. Each Game Object is responsible for keeping track of its graphical representation.
- The *Game Objects module* is a data repository containing information about game objects such as players, traps, monsters, platforms, etc.

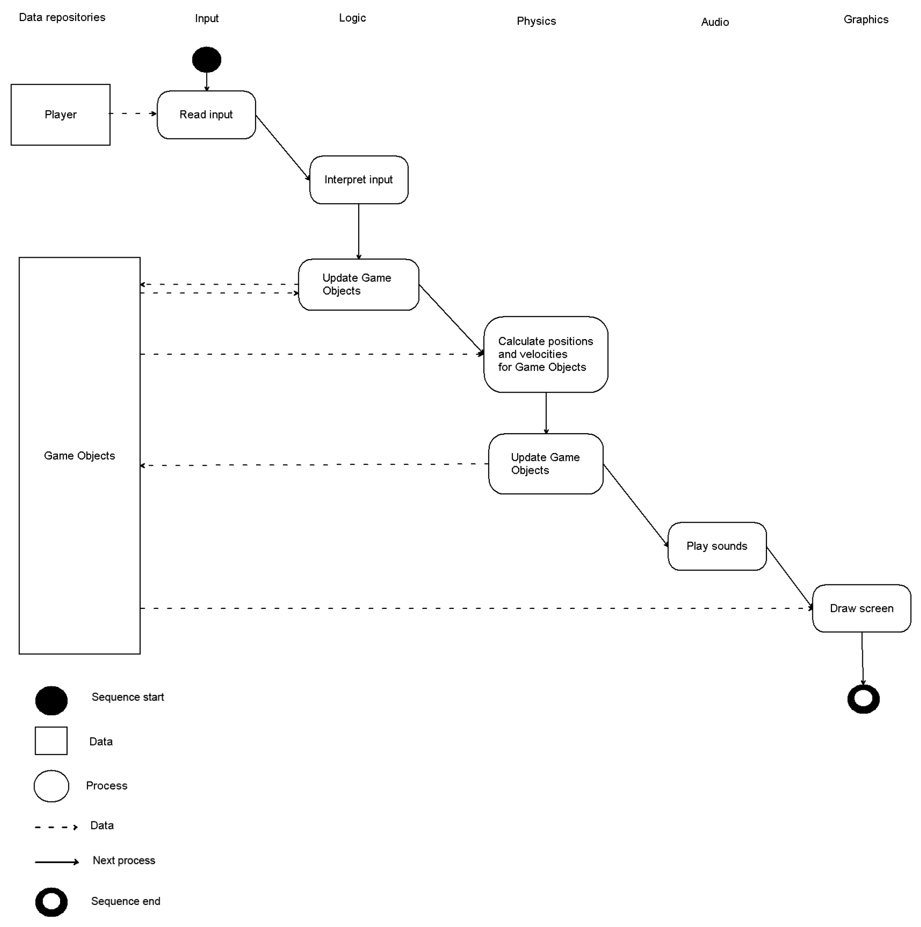
  Note, however, that the Game Objects module it not purely a data repository. It contains objects that have functions associated with them.

The game runs in a continous loop, in all states of the game. During every iteration of the loop the following things happen:

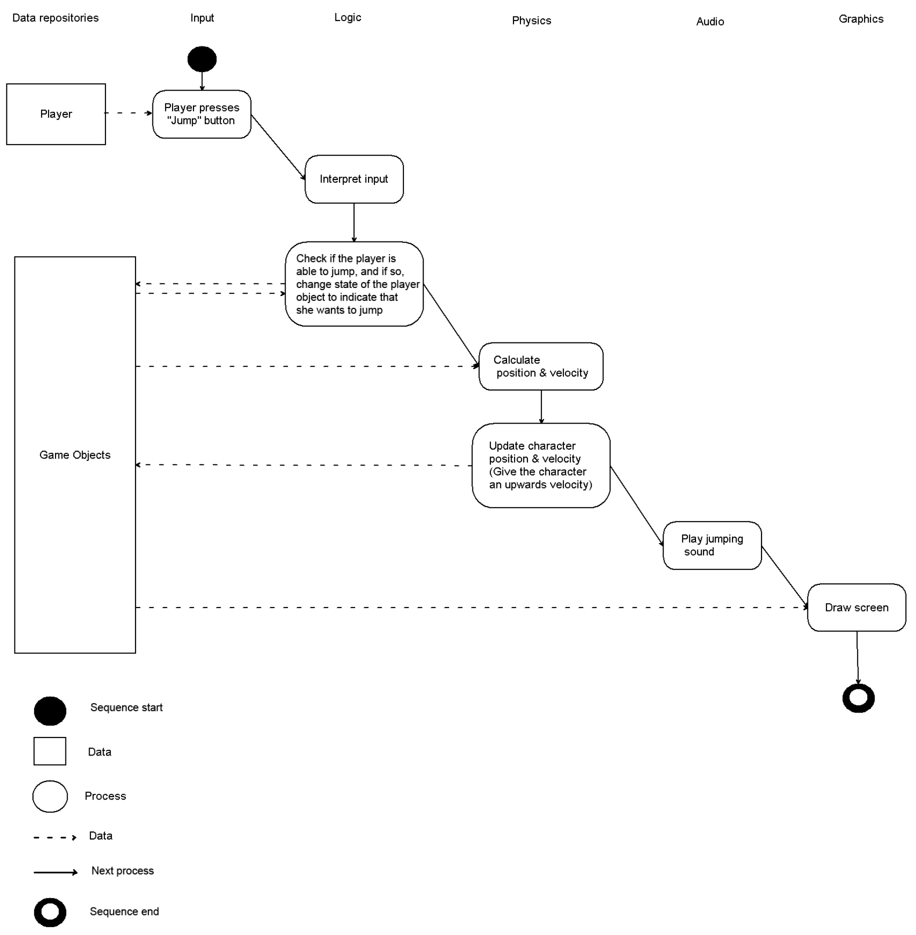
1. The Input module checks for player input. Player input is sent to the Logic module.
2. The Logic module interprets the player input, taking different actions depending on what state the game is in.
3. If the game is in the Game Running state, the Physics module is used to calculate positions and velocities for all game objects and to detect collisions between them.
4. After the Logic module has updated the game, the Audio module plays the appropriate sound effects and the Graphics module updates 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.



Data repositories
Input
Logic
Physics
Audio
Graphics
Player
Player presses "Jump" button
Interpret input
Check if the player is able to jump, and if so, change state of the player object to indicate that she wants to jump
Game Objects
Calculate position & velocity
Update character position & velocity (Give the character an upwards velocity)
Play jumping sound
Draw screen
Sequence start
Data
Process
Data
Next process
Sequence end

3 Design Considerations

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 Select Screen. At this screen each player confirms his 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 Select 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.

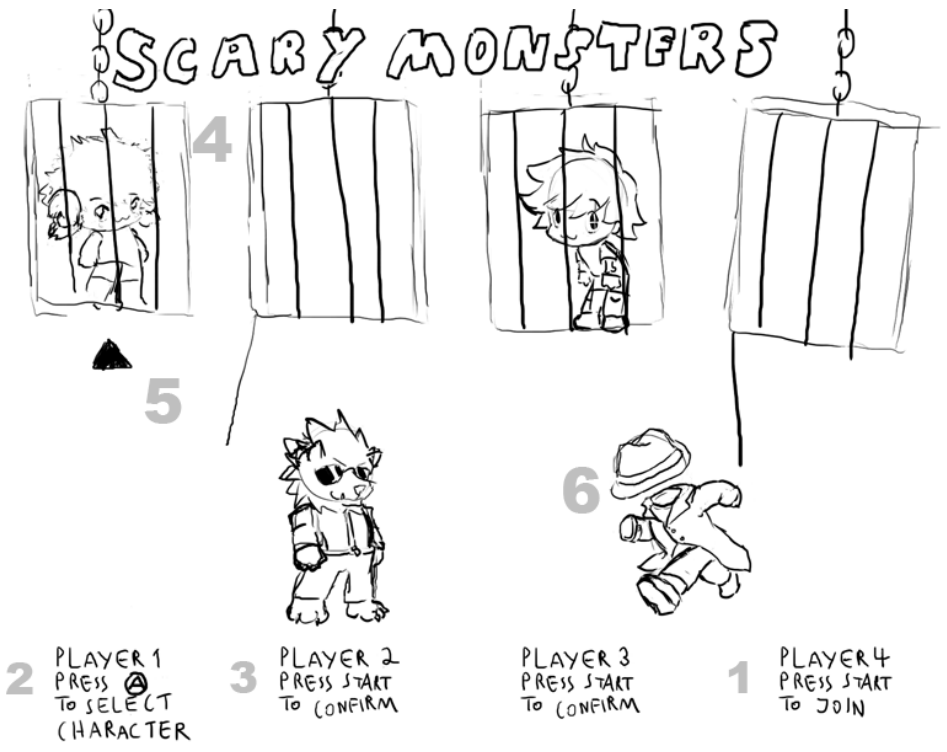
When the game stage is chosen the players are taken to the Game Screen. This is where the actual game is played.

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 Select Screen.

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

4.2 Pictures

4.2.1 Character Select Screen

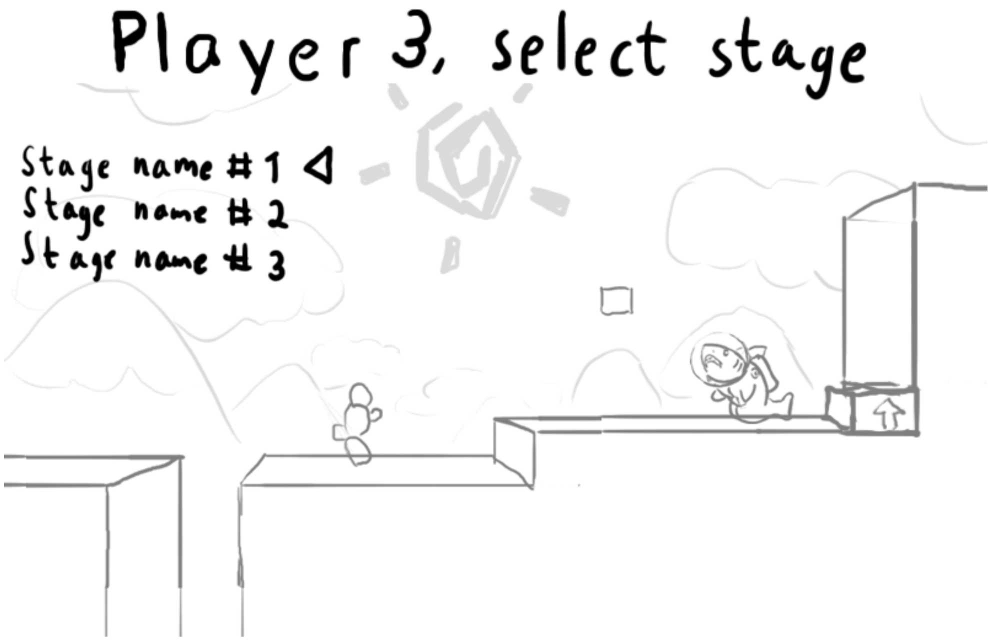


This is the character select screen. Players must first press the Start button in order to confirm their participation in the next race (1). Once a player has confirmed that she is to participate she can select one of the characters at the top of the screen (4). The players move selection arrows (5) 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. A player can not undo her choice of character. The arrows have different colors to make them distinguishable from each other.

Once selected, the character is let out of his cage (6). The player can now control her chosen characer as she would on the game stage. 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 the race begins.

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

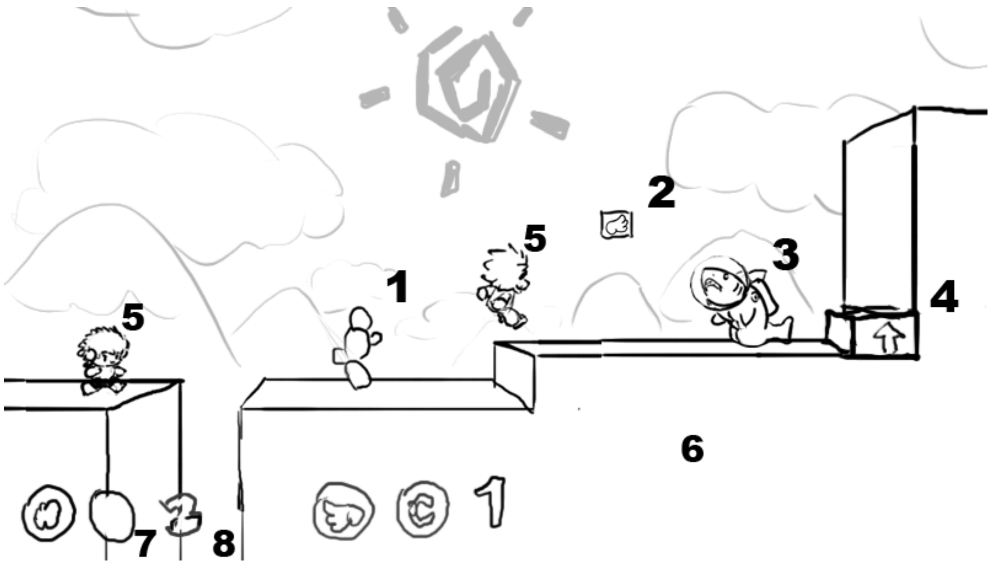
4.2.2 Game Stage Select Screen



This is the Game Stage Select 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



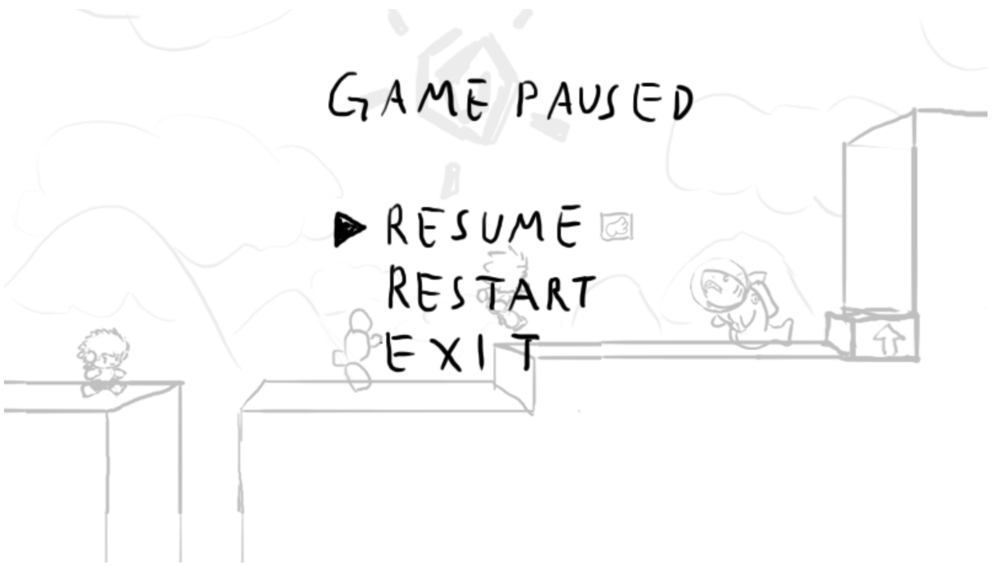
This is the game screen, where the actual game is played. The game elements (see Requirements Document section 4.1.2, requirements 1(b), 1(e) and 2(a)) shown here are:

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))

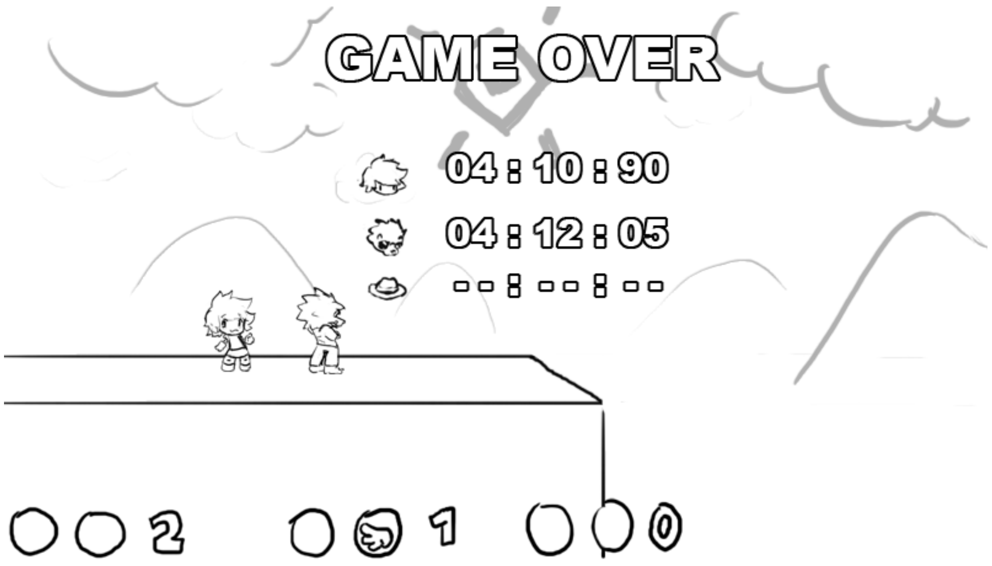
Notice also that the screen is not split up into smaller areas for each player. Instead the players all use the whole screen when playing. (See Requirements Document section 4.1.2, requirement 5(b))

4.2.4 Pause Screen



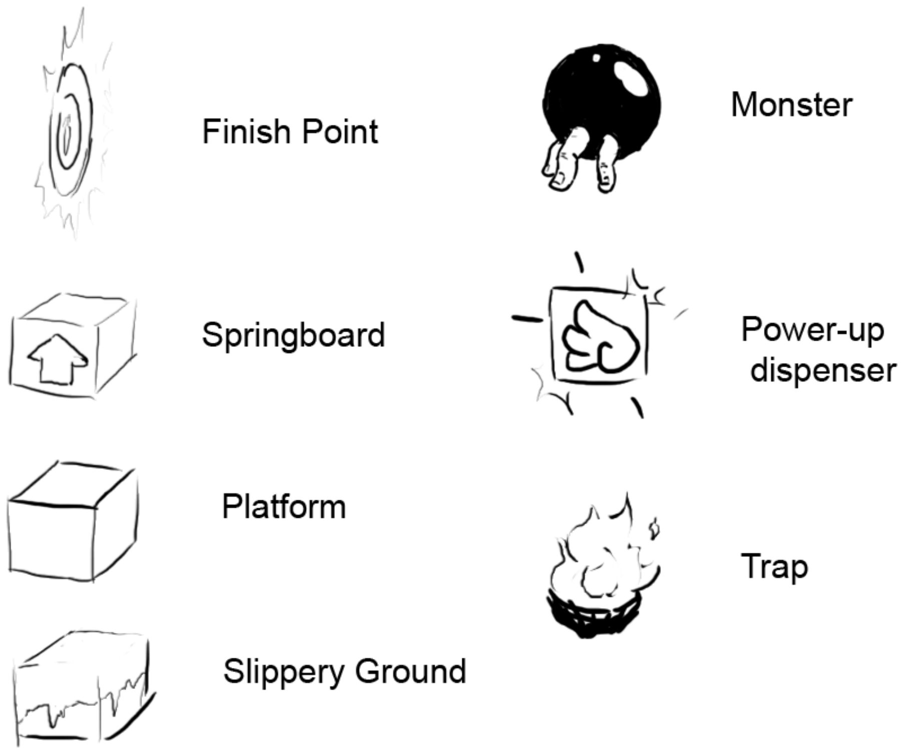
This is the Pause Screen. While a race is in progress, any player may press the Start button on her gamepad to pause the game. When the game is paused, everything in the race freezes and the players are presented with the pause menu. Only the player that paused the game may navigate the pause menu. (See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

4.2.5 Game Over Screen



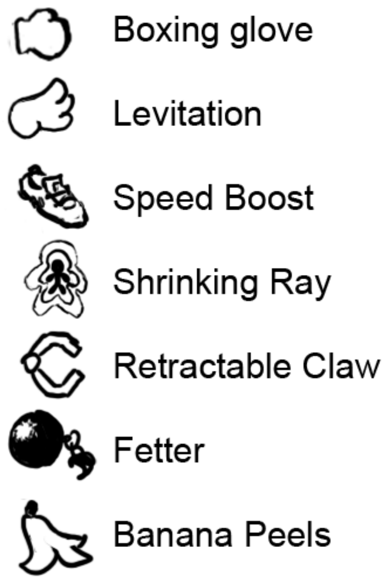
This is the Game Over Screen. When all players have either reached the finish point or lost all of their lives, the race ends and the Game Over Screen is shown. Here, the players are presented with their finish times. If a player lost all of her lives no finish time is shown for the player. (See Requirements 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 Game

Responsibilities:

- To start up the game and keep it running until the user wants to quit.
- Contains the main game loop—updates other modules.

Collaborators:

- Logic
- Physics
- Input
- Audio
- Graphics
- GameObjects

5.1.2 Logic

Responsibilities:

- Enforcing game rules

Collaborators:

- GameObjects

5.1.3 Input

Responsibilities:

- To retrieve gamepad button presses from XNA and interpret them into actions that the Logic class can understand.

Collaborators:

- GamePad (XNA)
- Logic

5.1.4 Audio

Responsibilities:

- To load music and sound effects from files.
- To play specified sounds when asked to.

Collaborators:

- AudioEngine (XNA)

5.1.5 GameObjects

Responsibilities:

- To load Game Stages from files.
- To store all objects in the game.

Collaborators:

- GameStage
- Player
- Trap
- MovingPlatform
- Monster
- SpringBoard
- BoxingGloveProjectile
- ClawProjectile
- ShrinkingRayProjectile
- BananaPeelProjectile
- FetterProjectile

5.1.6 GameStage

Responsibilities:

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

Collaborators:

- StartPoint
- FinishPoint

5.1.7 Graphics

Responsibilities:

- To draw everything

Collaborators:

- GraphicsDevice (XNA)
- GameObjects
- GameStage
- Player
- Trap
- MovingPlatform
- Monster
- SpringBoard
- BoxingGlove
- Claw
- ShrinkingRay
- BananaPeel
- Fetter
- Camera

5.1.8 Camera

Responsibilities:

- To store the camera positions

No collaborators.

5.1.9 GameObject (abstract class)

Responsibilities:

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

Collaborators:

- Geom (Farseer)

5.1.10 MovingPlatform (extends GameObject)

Responsibilities:

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

Collaborators:

- Path

5.1.11 PowerupDispenser (extends GameObject)

Responsibilities:

- Represents a power-up dispenser (see Requirments Document)

No collaborators.

5.1.12 FinishPoint (extends GameObject)

Responsibilities:

- Represents the Finish Point (see Requirments Document)

No collaborators.

5.1.13 StartingPoint (extends GameObject)

Responsibilities:

- Represents a Starting Point (see Requirments Document)

No collaborators.

5.1.14 Springboard (extends GameObject)

Responsibilities:

- Represents a Springboard (see Requirments Document)

No collaborators.

5.1.15 Trap (extends GameObject)

Responsibilities:

- Represents a trap (see Requirments Document).

No collaborators.

5.1.16 BoxingGlove (extends GameObject)

Responsibilities:

- Represents a boxing glove that is spawned when the Boxing Glove Power-up is applied (see Requirments Document).

No collaborators.

5.1.17 Fetter (extends GameObject)

Responsibilities:

- Represents a fetter that is spawned when the Fetter Power-up is applied (see Requirments Document).

No collaborators.

5.1.18 ShrinkingRay (extends GameObject)

Responsibilities:

- Represents a shrinking ray that is spawned when the Shrinking Ray Power-up is applied (see Requirments Document).

No collaborators.

5.1.19 Claw (extends GameObject)

Responsibilities:

- Represents a claw that is spawned when the Claw Power-up is applied (see Requirments Document).

No collaborators.

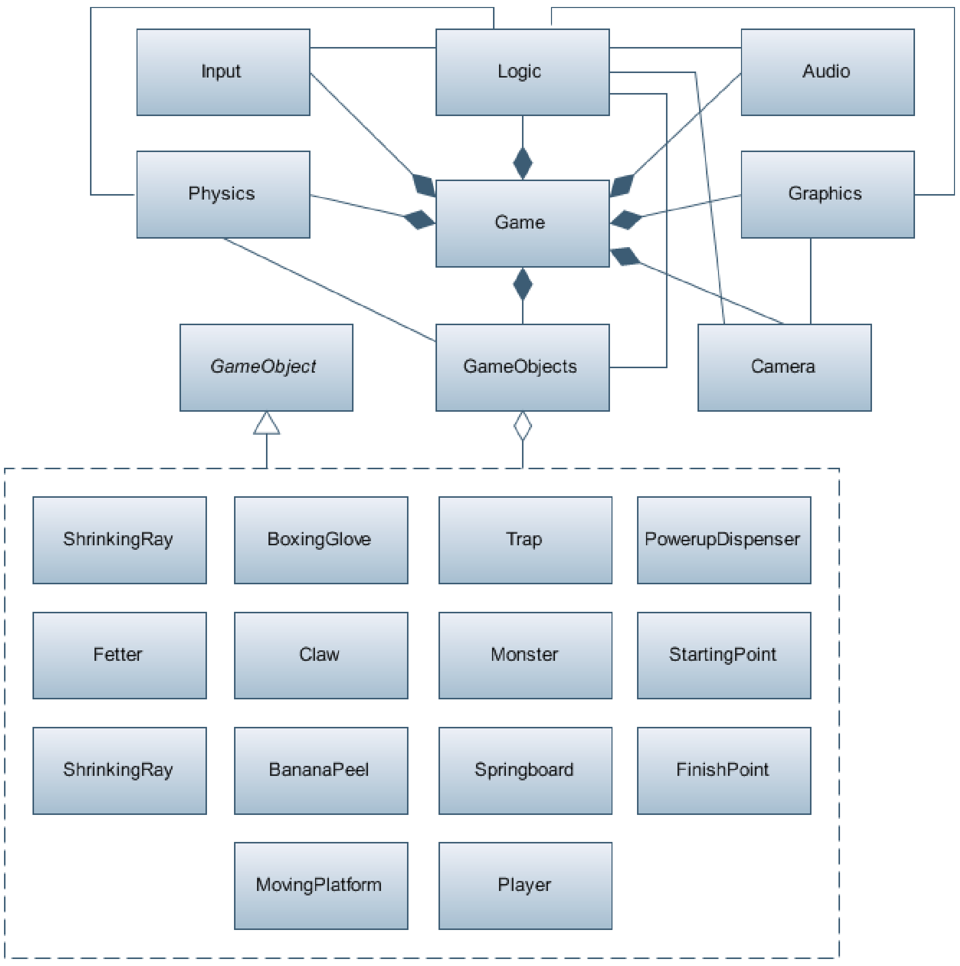
5.1.20 BananaPeel (extends GameObject)

Responsibilities:

- Represents a banana peel that is spawned when the Banana Peels Power-up is applied (see Requirments Document).

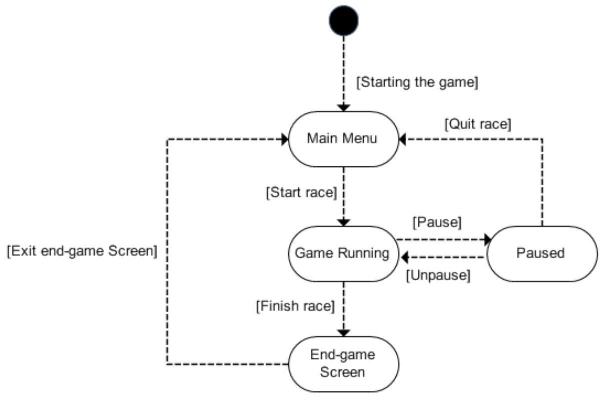
No collaborators.

5.2 Class Diagrams

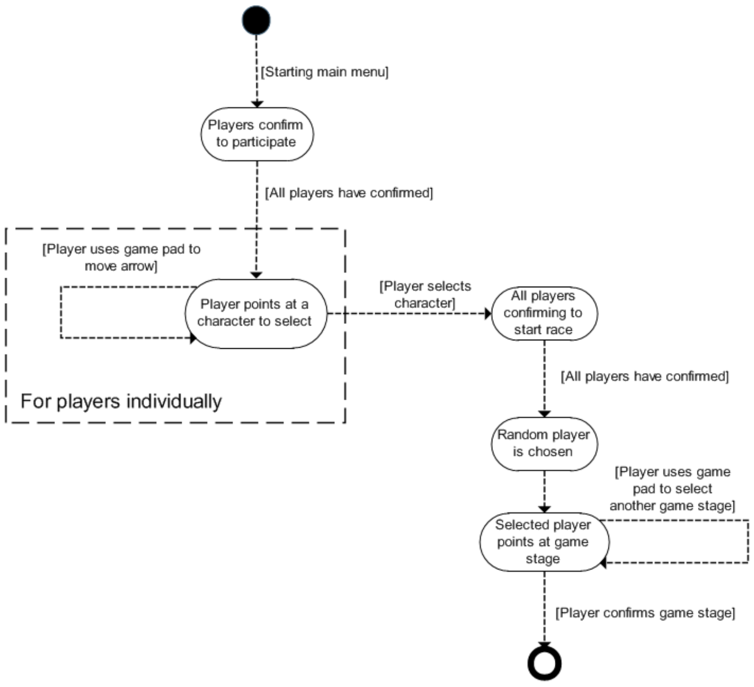


5.3 State Charts

5.3.1 Overall system

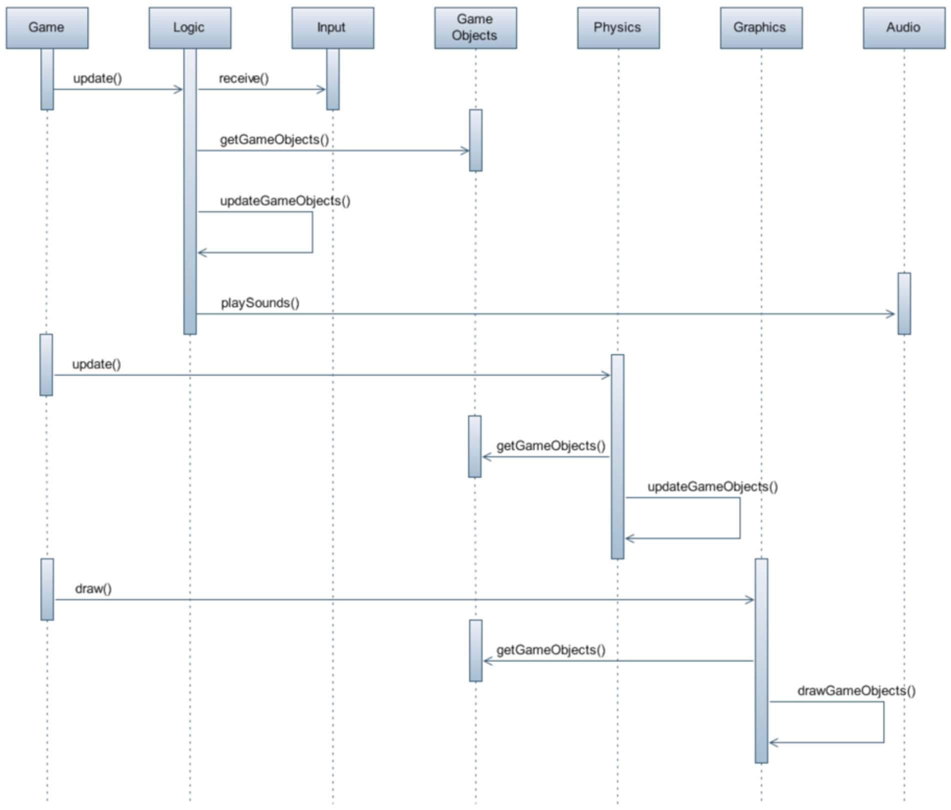


5.3.2 Main Menu


[Starting main menu]
Players confirm to participate
[All players have confirmed]
[Player uses game pad to move arrow]
Player points at a character to select
[Player selects character]
All players confirming to start race
For players individually
[All players have confirmed]
Random player is chosen
[Player uses game pad to select another game stage]
Selected player points at game stage
[Player confirms game stage]

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).

Fields:

- private Logic logic
- private Physics physics
- private Graphics graphics
- private Audio audio
- private Input input
- private GameObjects gameObjects
- private Camera camera

- private GraphicsDeviceManager device

Properties:

- public Logic Logic
- public Physics Physics
- public Graphics Graphics
- public Audio Audio
- public Input Input
- public GameObjects GameObjects
- public Camera Camera

Methods:

- ScaryMonsters()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: The constructor of the ScaryMonsters class. Instantiates Logic, Physics, Graphics, Audio, Input, GameObjects and Camera and stores a reference to each instance in the fields. Also instantiates the XNA class GraphicsDeviceManager.
  Data it accesses: None
  Preconditions: None
  Postconditions: Logic, Physics, Graphics, Audio, Input, GameObjects, Camera and GraphicsDeviceManager have all been instantiated and the instances stored in the fields of this instance of ScaryMonsters.
  Called by: ScaryMonsters.Main()
  Calls: None

- Update()

  Access modifier: protected
  Parameters: None
  Return value: None
  Description: This method calls the Update() methods on Logic, Physics and GameObjects.
  Data it accesses: None
  Preconditions: this.Initialize() must have been ran.
  Postconditions: Update() has been called on Logic, Physics and GameObjects.
  Called by: Game.Run()
  Calls: logic.Update(), physics.Update(), gameObjects.Update()

- Draw()

  Access modifier: protected
  Parameters: None
  Return value: None
  Description: Calls the Draw() method on graphics.
  Data it accesses: None
  Preconditions: this.Initialize() must have been ran.
  Postconditions: None
  Called by: Game.Run()
  Calls: graphics.Draw()

5.5.2 Logic

Fields:

- private ScaryMonsters sm

Properties: None
Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: This is where the main game loop takes place.
  Data it accesses: Data in GameObjects.
  Preconditions: None
  Postconditions: The game has been updated according to the game rules.
  Called by: Update() in the ScaryMonsters instance.
  Calls:

5.5.3 Input

Fields:

- private ScaryMonsters sm
- prive List<Controller>controllers

Properties: None
Methods:

- SomeonePressedStart()

  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

5.5.4 Controller

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

5.5.5 Physics

Fields:

- private ScaryMonsters sm

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

5.5.6 GameObjects

Fields:

- private ScaryMonsters sm
- private List<GameObjects>gameObjects
- private GameStage gameStage

Properties:

- public GameStage GameStage
- public List<GameObjects>GameObjects

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.

5.5.7 Graphics

Fields:

- private ScaryMonsters sm

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: Logic.Update()
  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

- DrawPauseMenu()

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

- DrawCharacterSelectScreen()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Draws the pause menu if the game is paused.
  Data it accesses: None
  Preconditions: The game is paused. The right settings on the GraphicsDevice has been set.
  Postconditions: The pause menu has been drawn this iteration of the game loop.
  Called by: Logic.Update()
  Calls: DrawGameObjects(), DrawHUD()

Fields:

- private Game game

Properties:
Methods:

- Name()

  Access modifier:
  Parameters:
  Return value:
  Description:
  Data it accesses:
  Preconditions:
  Postconditions:
  Called by:
  Calls:

5.5.8 Camera

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

5.5.9 GameObject

The GameObject class is an abstract class. It extends Geom from the Farseer physics engine.
Fields:

- private ScaryMonsters sm

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(Geom geom1, Geom geom2, ContactList contactList) (abstract method)

  Access modifier: public
  Parameters: geom1 - the Geom of this object, geom2 - the Geom of

the object this object collided with, contactList - a list of contacts in the collision
Return value: None
Description: Takes care of collisions with other objects in an approperiate way.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by:
Calls: None

5.5.10 ModelObject

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

- private Model model

Properties:

- public Model Model

Methods: None

5.5.11 SpriteObject

The SpriteObject class is an abstract class. It extends the GameObject class.
Fields: None

- private Texture2D sprite

Properties:

- public Texture2D Sprite

Methods: None

5.5.12 Effect

The Effect class is a wrapper containing information about a particular effect, from e.g a power-up
Fields:

- private String name
- private boolean active

- private int duration
- private int timeLeft

Properties: None

Methods:

- Update()

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

- Activate()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Activates the the effect.
  Data it accesses: active
  Preconditions: None
  Postconditions: None
  Called by: GameObject.Update()
  Calls: None

5.5.13 Player

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¿ activeEffects

Properties: None

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: activeEffects, controller, powerup1, powerup2
  Preconditions: None
  Postconditions: The player position and speed has been updated.
  Called by:
  Calls: None

- CollisionCallback(Geom geom1, Geom geom2, ContactList contactList)

  Access modifier: public
  Parameters: geom1 - the Geom of this object, geom2 - the Geom of the object this object collided with, contactList - a list of contacts in the collision
  Return value: boolean - true if collision is to be performed
  Description: Takes care of collisions with other objects in an approperiate 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
  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: Geom data field for speed
  Preconditions: None
  Postconditions: None
  Called by: Player.Update()
  Calls: None

5.5.14 Monster

The Monster class extends the ModelObject class.
Fields:

- private List¡Effect¿ activeEffects

Properties: None

Methods:

- Update()

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

- CollisionCallback(Geom geom1, Geom geom2, ContactList contactList)

  Access modifier: public
  Parameters: geom1 - the Geom of this object, geom2 - the Geom of the object this object collided with, contactList - a list of contacts in the collision
  Return value: boolean - true if collision is to be performed
  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

5.5.15 PowerupDispenser

The PowerupDispenser class extends the SpriteObject class
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(Geom geom1, Geom geom2, ContactList contactList)

  Access modifier: public
  Parameters: geom1 - the Geom of this object, geom2 - the Geom of the object this object collided with, contactList - a list of contacts in the collision
  Return value: boolean - true if collision is to be performed
  Description: Takes care of collisions with other objects in an approperiate way.
  Data it accesses: None
  Preconditions: None
  Postconditions: None
  Called by:
  Calls: None

5.5.16 Trap

The Trap class extends the ModelObject class
Fields:

- private boolean active
- private int duration
- private int timeLeft

Properties: None

Methods:

- Update()

  Access modifier: public
  Parameters: None
  Return value: None
  Description: Updates the Trap, deciding wether 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(Geom geom1, Geom geom2, ContactList contactList)

  Access modifier: public
  Parameters: geom1 - the Geom of this object, geom2 - the Geom of the object this object collided with, contactList - a list of contacts in

the collision
Return value: boolean - true if collision is to be performed
Description: Takes care of collisions with other objects in an approperiate way.
Data it accesses: None
Preconditions: None
Postconditions: None
Called by:
Calls: None

5.5.17 MovingPlatform

The MovingPlatform class represents a platform moving cyclically along a path in the Game Stage described by a collection of XNA.FrameWork.Points
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(Geom geom1, Geom geom2, ContactList contactList)

  Access modifier: public
  Parameters: geom1 - the Geom of this object, geom2 - the Geom of the object this object collided with, contactList - a list of contacts in the collision
  Return value: boolean - true if collision is to be performed
  Description: Takes care of collisions with other objects in an approperiate way.

Data it accesses: None
Preconditions: None
Postconditions: None
Called by:
Calls: None