Multiplayer Platform Game
Group 19

Martin Petterson
Oskar Kvist
Christoffer Ekeroth
Misael Berrios Salas

February 11, 2008

Contents

1 Introduction **2**

2 System Overview **2**
2.1 General Description 2
2.2 Overall Architecture Description 2
2.3 Detailed Architecture 5

3 Design Considerations **6**

4 Graphical User Interface **6**

5 Design Details **6**
5.1 CRC cards 6
5.1.1 Game 6
5.1.2 Logic 7
5.1.3 Input 7
5.1.4 Audio 7
5.1.5 GameObjects 8
5.1.6 GameStage 8
5.1.7 Graphics 9
5.1.8 Camera 9
5.1.9 GameObject (abstract class) 9
5.1.10 MovingPlatform (extends GameObject) 10
5.1.11 PowerupDispenser (extends GameObject) 10
5.1.12 FinishPoint (extends GameObject) 10
5.1.13 StartingPoint (extends GameObject) 10
5.1.14 Springboard (extends GameObject) 10
5.1.15 Trap (extends GameObject) 10
5.1.16 BoxingGlove (extends GameObject) 11
5.1.17 Fetter (extends GameObject) 11
5.1.18 ShrinkingRay (extends GameObject) 11
5.1.19 Claw (extends GameObject) 11
5.1.20 BananaPeel (extends GameObject) 11
5.2 Class Diagrams 12
5.3 State Charts 13
5.3.1 Overall system 13
5.3.2 Main Menu 13
5.4 Interaction Diagrams 14

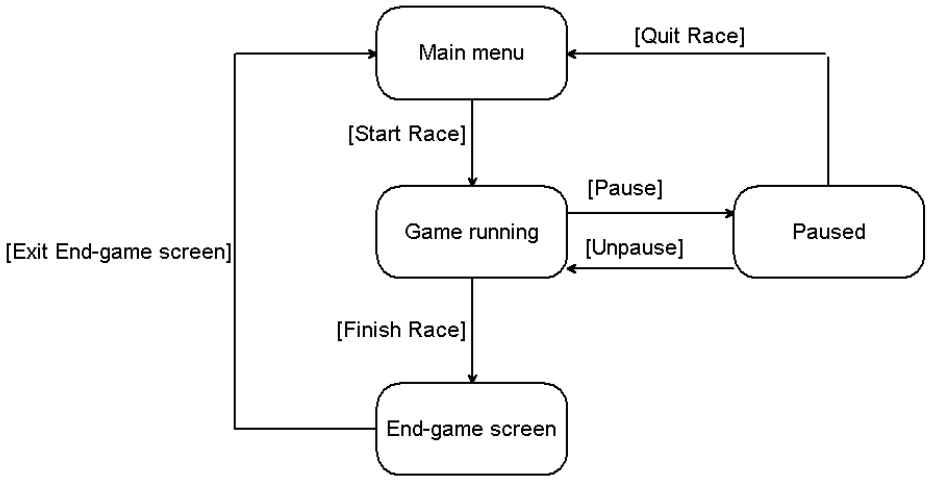
1 Introduction

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.

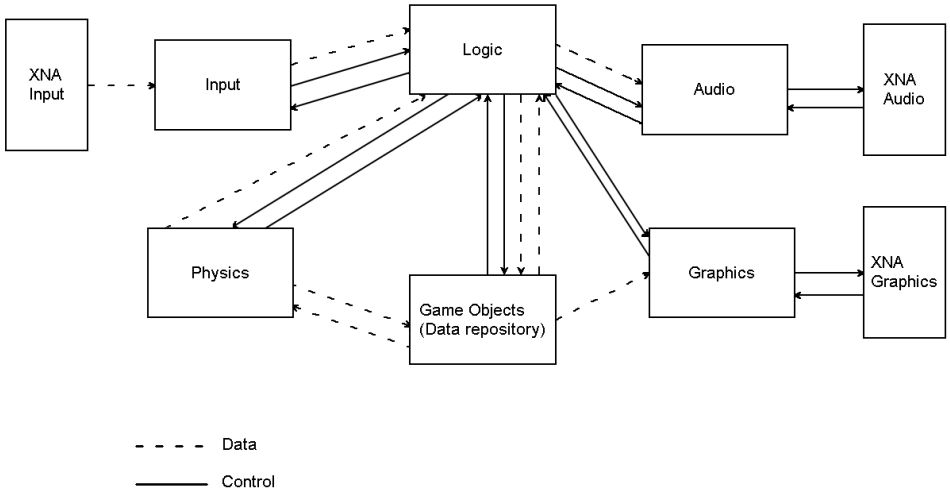


The game can be in any of four states at any given time. These states are:

- *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.

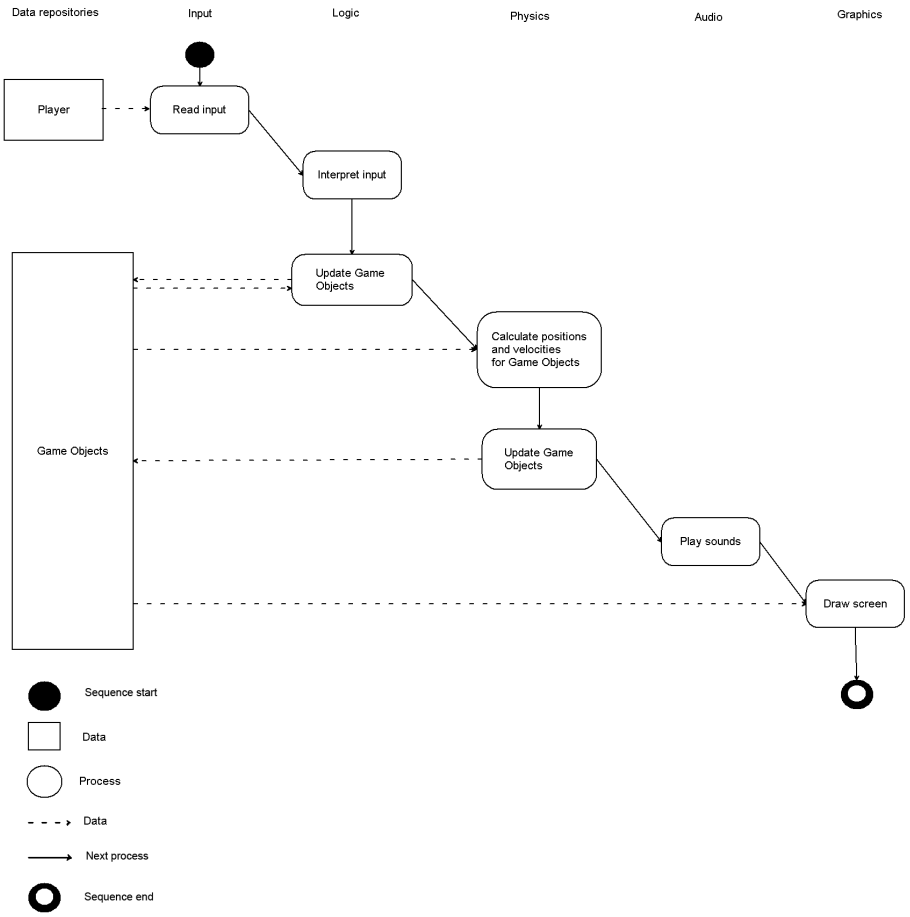
2. The Logic module interprets the player input, taking different actions depending on what state the game is in.

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.

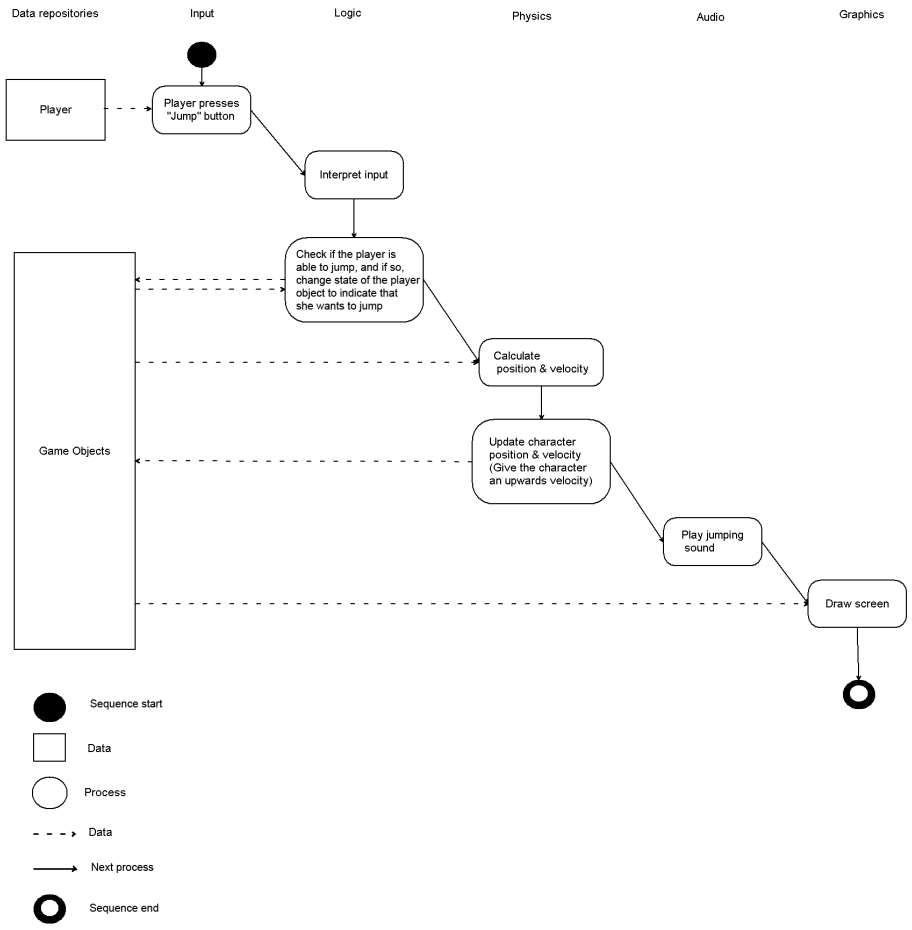
4. After the Logic module has updated the game, the Audio module plays the appropriate sound effects and the Graphics module updates the graphics on the screen.

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.



3 Design Considerations

4 Graphical User Interface

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.

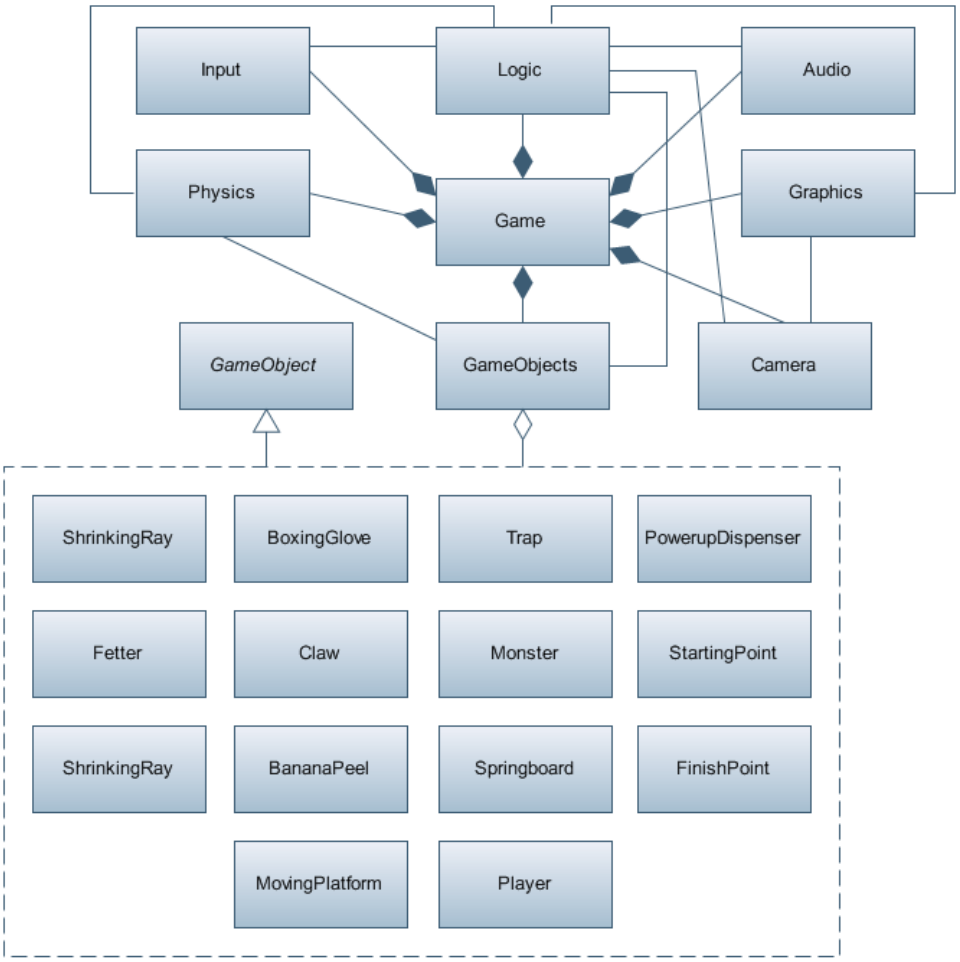
5.1.20 BananaPeel (extends GameObject)

Responsibilities:

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

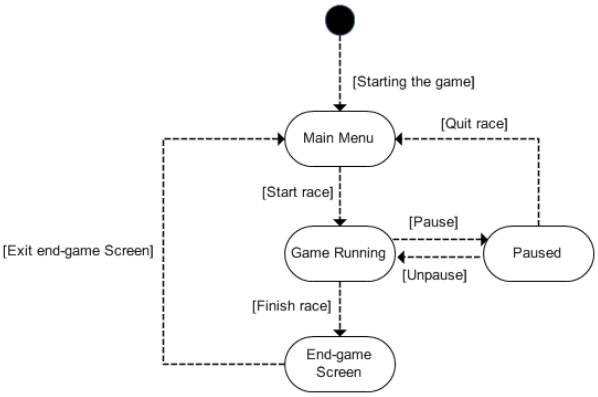
No collaborators.

5.2 Class Diagrams

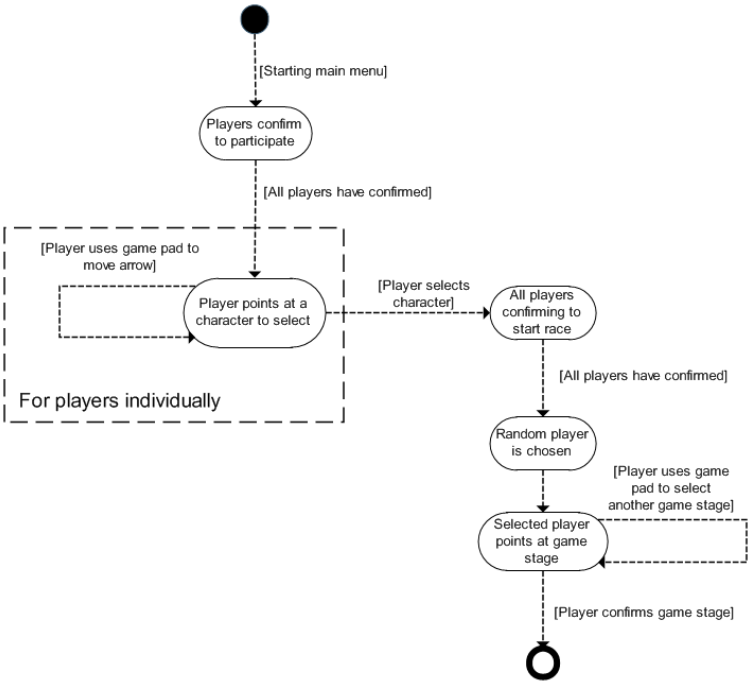


5.3 State Charts

5.3.1 Overall system



5.3.2 Main Menu



5.4 Interaction Diagrams