DOWN STREAM

Game Brigade

Ben Stevens, Brittney McFarlane, Omar Abdelaziz,
Dashiell Brown, Mauricio Moreyra, Theodore Billings

# Architecture Specification

## Class Responsibility Collaboration Tables

### Controllers

#### GDXRoot

**Description:** GDX Root is the cross-platform compatible root of the game.

**Justification:** As the root class, GDXRoot is essential for the game to exist.

| Responsibility | Collaboration |
|---|---|
| Initialize the game controller | DownstreamController |
| Initialize screen and canvas settings | GameCanvas |

#### WorldController

**Description:** This is the World Controller from Lab 4 and is largely unchanged.  Initializes the world and updates states such as complete and failure.  It adds and removes objects and references the canvas.  Its update method is abstract and we overwrite its draw method in Downstream Controller.

**Justification:** We need a high level world controller so we can extend it with our own significantly more specific game controller.

| Responsibility | Collaboration |
|---|---|
| Initialize the world | GameCanvas |
| Add and remove objects | Physics Objects |
| Get and set canvas and debug | Game Canvas |
| Get and set complete and failure | |

### DownstreamController

**Description:** This is the main controller for Downstream and is a subclass of World Controller. It loads assets and a level from LevelEditor. It updates the player by applying forces to control the koi fish. This controller also updates the game state and the camera state and movement.

**Justification:** We need a way to receive and process these inputs. The game state should be in respect to the fish (e.g. dead fish, alive fish) and the level.

| Responsibility | Collaboration |
|---|---|
| Initialize the level and controllers | LevelEditor, CameraController, InputController, AIController |
| Update objects and camera | Physics objects, CameraController, GameCanvas, CollisionController |
| Load assets and level geometry | LevelEditor |
| Get and Set game state (i.e. goal, death) | Physics objects |

### LevelEditor

**Description:** This controller reads from a JSON file storing a level and places the corresponding objects into the game world. In addition, it saves a level to a JSON upon player request.

**Justification:** Levels are saved in JSON format, and we need a way to translate that information into the game world. Because this class is working with JSON files and building levels, it also makes sense for it to control saving and loading, which are also done by interacting with JSON files. The external GSON library is used for these operations and is described at the end of this document.

| Responsibility | Collaboration |
|---|---|
| Load game world from .json file | LevelModel, TetherModel, EnemyModel, TerrainModel, WhirlpoolModel |
| Save game state | |
| Load game from save state | |

## AIController

**Description:** This controls the Enemy Fish AI. It uses a state machine to update an enemy's chasing state, and uses A* to set its path.

**Justification:** We need to create more complex challenges. This will mainly change the gameplay depending on the conditions for when to follow the player fish or not, which gets harder on different levels.

| Responsibility | Collaboration |
|---|---|
| Get and set the state of the enemy AI | EnemyModel |
| Set search path of AI | |

## InputController

**Description:** The input controller detects input from the player and sets flags based on the actions the player takes.

**Justification:** This controller is needed for the player to be able to interact with the game.

| Responsibility | Collaboration |
|---|---|
| Read player input | |
| Return current input state | |

## CollisionController

**Description:** This controller updates object states based on collisions between the player and other objects.  Currently, contact with an enemy or terrain model is a death condition while a tether is either orbited or passed.  Whirlpools redirect a player's direction.

**Justification:** We will have several different kinds of collisions between different objects, so we should have our own controller to handle these cases.

| Responsibility | Collaboration |
|---|---|
| Update player and tether | PlayerModel, TetherModel |
| Update player and enemy | PlayerModel, EnemyModel |
| Update player and terrain | PlayerModel, TerrainModel |
| Update player and whirlpool | PlayerModel, WhirlpoolModel |
| Update player and rocks | PlayerModel, RockModel |

## LoadingMode

**Description:** Loading mode creates a loading screen while minimally pre-loading assets.

**Justification:** Loading assets beforehand, rather than in game, prevents major loading times between levels.

| Responsibility | Collaboration |
|---|---|
| Draw loading screen | GameCanvas |
| Load all game assets | |

### CameraController

**Description:** The camera controller updates camera movement and is responsible for ensuring that the camera follows the player and feels natural. For example, when the player is tethered, the camera centers on the tether and zooms out. When the player is moving between tethers the camera follows the player.

**Justification:** Based on the current game state, the camera will do different things, so a class needs to be responsible for updating it.

| Responsibility | Collaboration |
|---|---|
| Translate the camera | GameCanvas |
| Zoom in and out | GameCanvas |

---

## Models

### PlayerModel

**Description:** An instance of PlayerModel gets and sets player states.

**Justification:** This model allows us to adjust the state of the fish, whether or not it reached the goal, and its force that is applied with respect to the tether.

| Responsibility | Collaboration |
|---|---|
| Get and set all state information about the player fish | |
| Get and set tether forces | |
| Apply forces to the fish | |
| Gets tangent and intersection information | |
| Draw the player to the screen | GameCanvas |

## EnemyModel

**Description:** An instance of EnemyModel gets and sets enemy states.

**Justification:** We need a model that is responsible for getting and setting the enemy fish state to add challenges to our game.

| Responsibility | Collaboration |
|---|---|
| Get and set the patrol path and the current location of the enemy | |
| Draw the enemy to the screen | GameCanvas |

## TetherModel

**Description:** This model gets and sets all tether states.

**Justification:** Different tethers can contain different properties. For example, we need to be able to adjust the progress a fish has made when circling a tether. It is easiest to update whether or not a tether has been orbited through this model.

| Responsibility | Collaboration |
|---|---|
| Get and set the location and orbital radius | |
| Draw the tether to the screen | GameCanvas |

## WhirlpoolModel

**Description:** This model gets and sets whirlpool states.

**Justification:** Forces applied to the player will need to be updated depending on different properties of the whirlpool.

| Responsibility | Collaboration |
|---|---|
| Get and set the location and the spinning force of the whirlpool | |
| Draw the whirlpool to the screen | GameCanvas |

## TerrainModel

**Description:** This model gets and sets states of land masses and other impassable terrain.

**Justification:** Players can collide with land, so there needs to be a class to store information about it.

| Responsibility | Collaboration |
|---|---|
| Get and set the location and shape of the terrain | |
| Draw the terrain to the screen | GameCanvas |

## LevelModel

**Description:** This model stores information about a level; i.e. where are tethers, the player, enemies, terrain, etc.

**Justification:** We need to store all of the information taken from a JSON file and eventually turn it into a Java class for ease of use and modifiability.

| Responsibility | Collaboration |
|---|---|
| Store tethers, player, enemies, terrain | TetherModel, PlayerModel, EnemyModel, TerrainModel, WhirlpoolModel |

# View

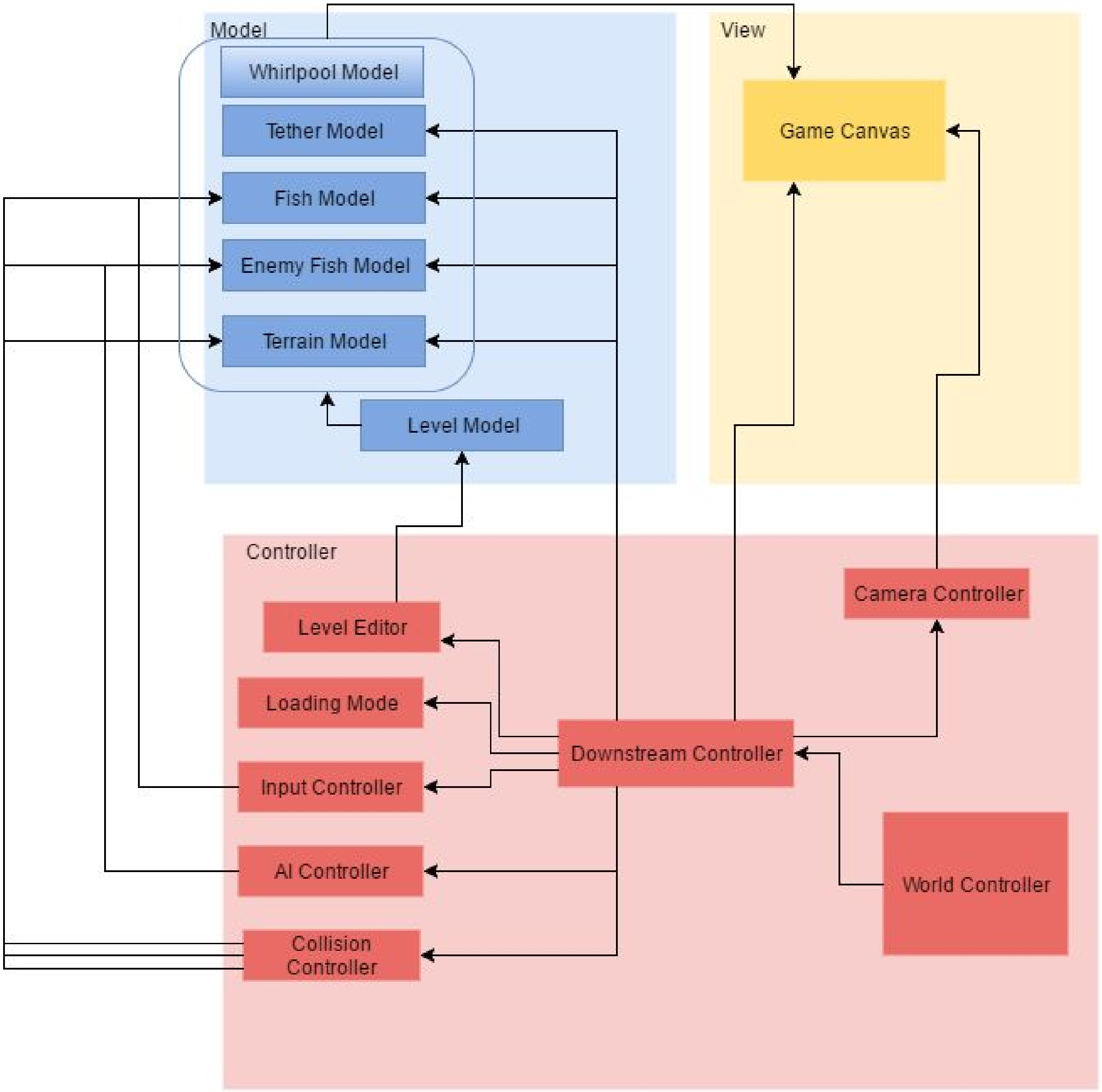
## GameCanvas

**Description:** This is the canvas for the game. It draws all textures.

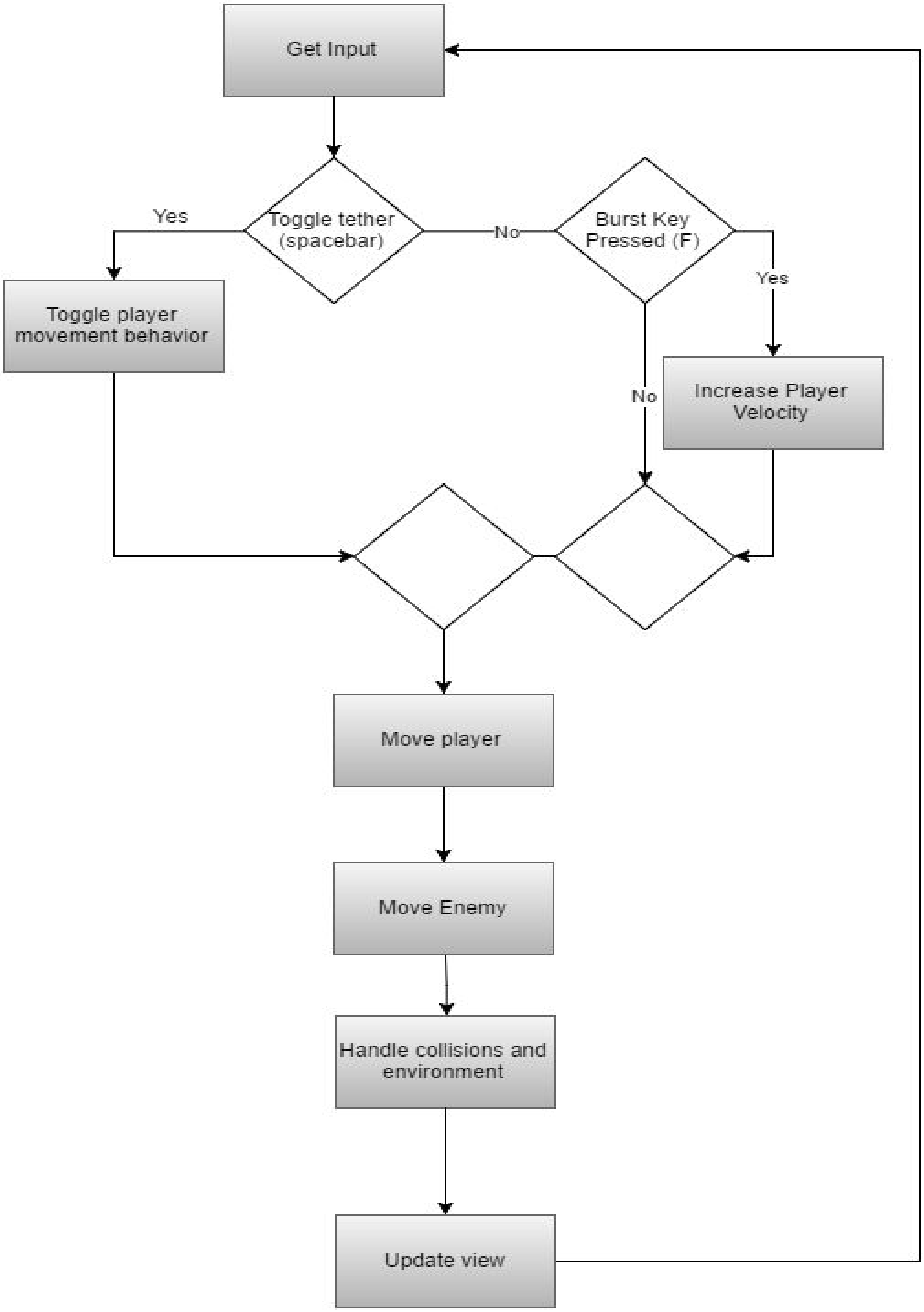
**Justification:** All other information about levels is elsewhere.  We only need to consolidate drawing textures to this class.

| **Responsibility** | **Collaboration** |
|---|---|
| Draw all object textures | |

# Dependency Diagram



# Activity Diagram



# Data Representation Model

## Save Game File

**File Format:** JSON format will be the data format for our Save Game File.
**Information:** The following information will be contained within the Save Game File as a JSON object under a top level: the level name; if the level was completed; if the level was unlocked.
**Storage Format**:

```
{
        levels:[
                {       levelname: “level1”
                        complete: false
                        unlocked: true
                }
                {       levelname: “level2”
                        complete: false
                        unlocked: true
                }
                {       levelname: “level3”
                        complete: false
                        unlocked: true
                }
        ]
}
```

## Level File

**File Format:** JSON format will be the data format for our Level File.
Information: The following information will be contained within the Level File: the vector starting position of the player fish model; Enemy Fish starting position, and an arrayList of vector positions that the Enemy fish travels; all tether Vector positions; a list of wall positions; game size.
**Storage Format:**

```
{
                    dimensions:{
                x: 10
                y: 10
        }
        fish:{
                x: 10
                y: 10
        }
        tethers:[
                {       type: lilypad
                        position: {
                                x: 10
                                y: 10
                        }
                },
                {       type: lantern
                        position: {
                                x: 10
                                y: 10
                        }
                }

        ]
        enemies:[
                {       type: fast
                        position: {
                                x: 10
                                y: 10
                        }
                        state: rest
                },
                {       type: slow
                        position: {
                                x: 10
                                y: 10
                        }
                        state: attack
                }

        ]
        walls:[
                {       coordinates: [10, 20, 30, 40, 50]
                        texture: "wall.jpg"
                }
                {       coordinates: [20, 30, 40, 50]
                        texture: "wall1.jpg"
                }
        ]
}
```

# Third Party Libraries

## Third Party Libraries

### Box2D

Box2D will serve as Downstream's physics engine. Box2D allows us to use pre-defined physics objects to simplify the movement system in our game. By using Box2D's structures, we have access to sensor and object collisions.

## GSON

This library is used for converting back and forth between JSON files and Java objects.  We use its fromJson() and toJson() methods.