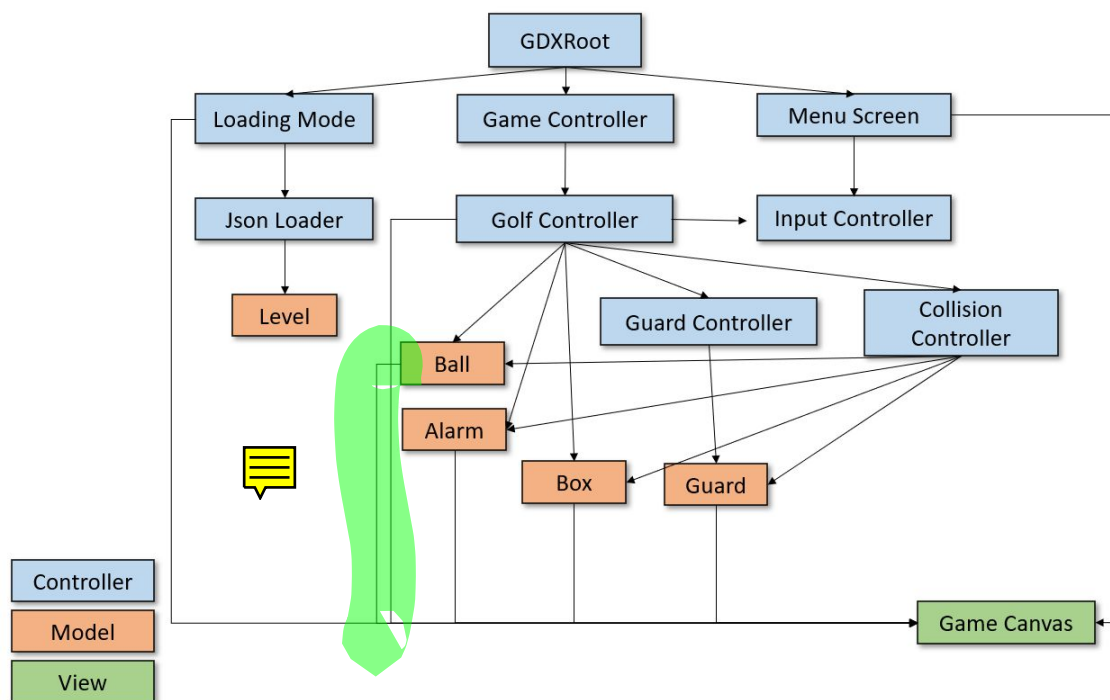# Architecture Specification
Parole in One

Waypoint. Isabel Selin, Yuxiang Yu, Courtney Manbeck, Lucien Eckert, Betsy Vasquez Valerio, Kevin Klaben, Tony Qin, Barry Wang

Dependency Diagram

Class-Responsibility Collaboration Divisions

Controller

GDXRoot

➔ Description: This class is the root class of the game. It will initialize the golf controller, loading screen controller, and menu screen controller. It will handle the transition between these 3 states.
➔ Justification: This class is the root of the game.

Responsibilities	Collaborators
Initialize controllers	LoadingMode, MenuScreen, GameController
Loading assets	Loading Mode
Transition between states	LoadingMode, MenuScreen, GameController
Exit game	-

MenuScreen

➔ Description: This class handles the main menu screen as well as possibly the option menu screen if that is to be implemented.
➔ Justification: This controller is needed to handle most interactions when not actively playing a level.

Responsibilities	Collaborators
Draw menu	GameCanvas
Choose level	InputController
Provide status flag	GDXRoot

LoadingMode

- ➔ Description: This class handles display and interaction during and immediately after asset loading.
- ➔ Justification: This class handles asset pre-loading.

Responsibilities	Collaborators
Load assets	JsonLoader
Draw animation	GameCanvas
<mark>Provide level objects</mark>	-

JsonLoader

- ➔ Description: This class parses the level JSON file into a usable format, currently envisioned as a level model.
- ➔ Justification: While the golf controller can do this as well, we've temporarily decided to split this off as a separate class so the golf controller isn't overloaded.

Responsibilities	Collaborators
Create Level Objects	LevelModel
<mark>I/O</mark>	-

GameController

- ➔ Description: The game controller will create new instances of golf controllers for each level as the levels are requested. It'll also keep track of states related to gameplay such as victory and loss.
- ➔ Justification: This controller is needed to create and handle the golf controller currently running and pause it if needed.

Responsibilities	Collaborators
Create level worlds	GolfController, LevelModel
Provide status flag	GDXRoot, <mark>GolfController</mark>

GolfController

- ➔ Description: The main controller for each level, keeps track of positions of elements of a level such as walls, guards, and alarms.
- ➔ Justification: This controller is needed to abstract and run the main game loop.

Responsibilities	Collaborators
Receive winning/losing flags	GameController
Provide status flag	GameController
Update game screen	InputController, CollisionController
Draw game screen	GameCanvas

GuardController

- ➔ Description: This is the AI controller that will change guard states and give guards instructions to move. Currently it also handles raycasting detection from each guard).
- ➔ Justification: A controller is necessary to handle state changes ane pathfinding in guards,

Responsibilities	Collaborators
Pathfinding	GolfController
Determine guard state changes	GolfController
Set guard movement	GuardModel

CollisionController

→ Description: This is the controller that will determine the correct behaviour of each collision that occurs.
→ Justification: A collision controller is needed since we have slightly complex interactions (such as alarm activation and deactivation)

Responsibilities	Collaborators
Resolve collisions	BallModel, GuardModel, AlarmModel, BoxModel
Update winning/losing status	GolfController

InputController

→ Description: This is the controller that reads player input and allows other controllers to use these inputs.
→ Justification: An input controller is needed to allow the players to play the game.

Responsibilities	Collaborators
Read input	-

Model

AlarmModel

➔ Description: This model keeps track of the state of an alarm, mainly whether or not it is activated.

➔ Justification: This is a model for the alarm environmental object.

Responsibilities	Collaborators
Provide status	-

BallModel

➔ Description: This model keeps track of the ball's physical properties, including its body, shape, and fixture.

➔ Justification: This is a model for the player-controlled ball object.

Responsibilities	Collaborators
Receive force	-

GuardModel

➔ Description: This model keeps track of properties of guards. This includes their patrol paths, starting locations, and sight radius and distance.

➔ Justification: This is a model for the guard enemy.

Responsibilities	Collaborators
Receive velocities	-

BoxModel

➔ Description: This model keeps track of the physical properties of boxes, including its body, shape, and fixture.
➔ Justification: This is a model for the box environmental object.

Responsibilities	Collaborators
Receive force	-

LevelModel

➔ Description: This model carries information about a level. This includes the placement of walls, position of alarms and boxes, position and patrol paths of guards, and the ball starting and goal positions.
➔ Justification: We plan to convert the level JSON into an intermediate format before creating the actual objects so we don't have to create all the objects for all the levels before they are needed. If we determine that the cost to create the level objects are low, then this model may be deprecated.

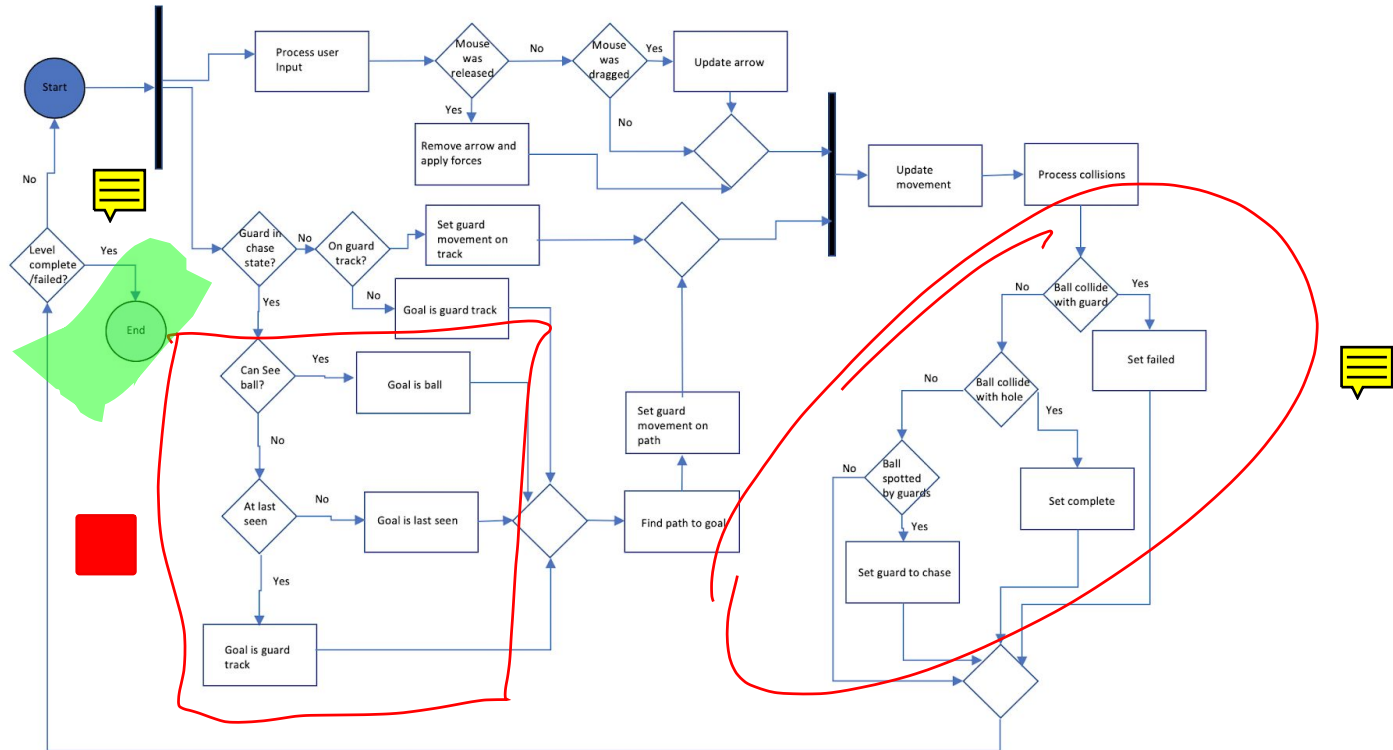| Responsibilities	Collaborators
-	-

View

GameCanvas

➔ Description: The game canvas draws objects onto the screen using their positions as well as any other transformations that may be applied to each object.
➔ Justification: Game canvas must exist to facilitate the drawing of objects to the screen.

Responsibilities	Collaborators
Draw Textures	-

Activity Diagram

Start

Process user Input

Mouse was released

Mouse was dragged

Update arrow

No → Yes

Remove arrow and apply forces

Update movement

Process collisions

Level complete /failed?

No / Yes

End

Guard in chase state?

On guard track?

Set guard movement on track

No / Yes

Goal is guard track

Can See ball?

Yes

Goal is ball

No

At last seen

No

Goal is last seen

Yes

Goal is guard track

Set guard movement on path

Find path to goal

Ball collide with guard

No / Yes

Set failed

Ball collide with hole

No / Yes

Set complete

Ball spotted by guards

No / Yes

Set guard to chase

Data Representation Model

Saved Game File

Overview

The game will store saved data in a JSON file. This JSON will contain the state of the game (LoadingMode, MenuScreen, GameController) and what level the player has reached. The keys of the JSON will be "gamestate" and "level".

Example

```
{"gamestate": "MenuScreen", "level": 3 }
```

Level File

Overview

Each level will be stored in JSON file format. The levels will each contain the following information: level number, placement of walls, position of alarms and boxes, position and patrol paths of guards, and the ball starting and goal positions.

level

This section of the JSON will be an integer denoting the number of the level the JSON is representing.

walls

This section of the JSON will be a list of screen coordinate locations of where walls should be placed.

alarms

This section of the JSON will be a list of screen coordinate locations of where alarms should be placed.

boxes

This section of the JSON will be a list of screen coordinate locations of where boxes should be placed.

guard_positions

This section of the JSON will be a list of screen coordinate locations of where guards should be placed.

guard_patrols

This section of the JSON will be a list of lists of screen coordinate locations. The length of this list will be the same as the length of the list in the guard_positions section. guard_patrols[i] corresponds to the patrol path of the guard in guard_positions[i].

ball_start

This section of the JSON will be a tuple of screen coordinate locations of where the ball should be when the level is initialized.

ball_goal

This section of the JSON will be a tuple of screen coordinate locations of where the ball should be to reach the goal state.

Example

```
{ "level": 1,
"walls" : [[(0,0),(10, 0), (0, 10), (10,
10)],[(50,50),(60,70),(70,50)]],
"alarms" : [(20, 10)],
"boxes" : [ (70, 30)],
"guard_positions" : [(50, 20)],
"guard_patrols" : [[(50,30),(50,20), (50,10)]],
"ball_start" : (0,0),
"ball_goal" : (100, 50) }
```

3152: Assignment 9 (Architecture Document)

Assessor: Walker White

SA: Strongly Agree   **D**: Disagree

A: Agree   **SD**: Strongly Disagree

	SD	D	A	SA
The dependency diagram is an easy-to-follow summary of the architecture.				
• The dependency directions are clear from the diagram.				✓
• The information flow can be discerned from the diagram.		✓		
• The diagram does not have too many overlapping edges.				✓
• The diagram does not have any cycles or improperly coupled classes.		✓		
The CRC tables adequately describe the class structure.				
• The class choices are sensible and properly organized.			✓	
• The architecture can support serialized levels.	✓			
• The collaborators appear correct for each responsibility.		✓		
• There are no obvious missing responsibilities.		✓		
• There are no improper responsibilities (e.g. controller code in models).			✓	
• The responsibility descriptions are active and appropriate.		✓		
The activity diagram clearly describes the flow of a single frame.				
• The activities are described at the appropriate resolution.		✓		
• The activities are consistent with the CRC tables.			✓	
• The diagram uses parallelism when order is not necessary.			✓	
• The diagram follows proper UML conventions.				✓
The serialization format supports external tool support.				
• The file format appears to store the necessary information.				✓
• The high-level attributes are all clearly identified and described.				✓
• The format has an illustrative, but brief, example.				✓
The document is clear and well-written.				
• The document contains no spelling errors.				✓
• The document contains no grammatical errors.				✓
• The sections are properly organized.				✓
• The document adheres to the course writing guidelines.		✓		