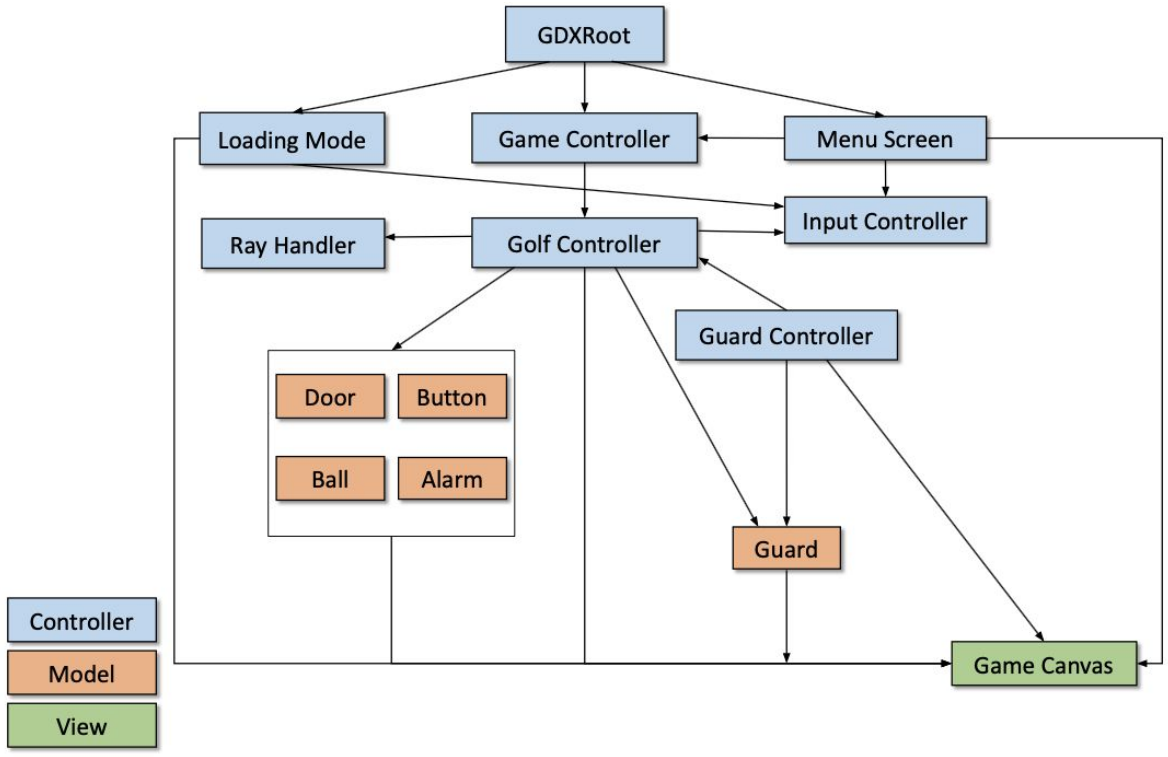
Architecture Specification
Parole in One

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 game controller, loading screen, and menu screen. It will handle the transition between these 3 screens.

Justification: -

Responsibilities	Collaborators
Initialize controllers	LoadingMode, MenuScreen, GameController
Listen for screen transition cues	LoadingMode, MenuScreen, GameController

MenuScreen

Description: This class presents the main menu screen for choosing levels and settings

Justification: This controller is needed to handle most interactions when not actively playing a level.

Responsibilities	Collaborators
Load menu screen assets	AssetManager
Draw menu screen	GameCanvas
Get information about button presses	InputController
Push level number currently selected	GameController
Push exit code	ScreenListener

LoadingMode

Description: This class handles display and interaction during and immediately after asset loading.

Justification: This class makes time for and ensures assets to pre-load.

Responsibilities	Collaborators
Load loading-screen assets	AssetManager
Get progress of loading remaining game assets	AssetManager
Display progress of loading remaining game assets	GameCanvas
Get information about button presses	InputController
Push exit code	ScreenListener

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 levels completed and scores of each level. The game controller is also used to parse JSON files.

Justification: This controller allows us to detect whether the level data inputted is corrupt. This controller is also required to keep track of game information that persists across levels.

Responsibilities	Collaborators
Parse level data from JSON file	-
Instantiate levels using level data	GolfController
Listen for pause and exit level requests	GolfController
Push exit code to ScreenListener	ScreenListener

GolfController

Description: Each golf controller controls one level in the game. This controller contains information about its level, such as the position of game elements. The controller also converts player input into actual gameplay, and handles all collisions between game objects.

Justification: This controller is needed to run the main game loop, and allows for game elements to interact with each other.

Responsibilities	Collaborators
Load game assets	AssetManager
Assign textures to models	All model classes
Create game objects and add them to the world	All model classes
Create and attach light elements to game assets	RayHandler
Get information about shot to set ball movement	InputController, BallModel
Push information about contacts between elements	CollisionController
Draw backgrounds, walls, and HUDs	GameCanvas
Draw Box2d lights effects	RayHandler
Set victory or loss field in BallModel if it touches the goal or a guard respectively	BallModel
Set alarm to activated or deactivated if it touches the ball or a guard respectively	AlarmModel
Change button and door state if pressed by the ball	ButtonModel, DoorModel
Change the ball velocity if it touches a wall	BallModel

GuardController

Description: This is the AI controller that will change guard states and give guards instructions to move. It also handles ball detection from each guard. Guardcontroller obtains data from GolfController to handle state changes and movement planning.

Justification: A controller is necessary to handle state changes and pathfinding of guards.

Responsibilities	Collaborators
Change guard state if applicable	GolfController
Plan guard movement	GolfController
Set guard movement	GuardModel
Draw guard patrol indicators	GameCanvas

InputController

Description: This is the controller that converts player input into boolean flags and integer values to allow other elements of the game to not deal with specific user inputs.

Justification: This class abstracts out information about user input to one location and stores all relevant user input for each controller to get information from. This class also converts inputs from different devices into a standardized format.

Responsibilities	Collaborators
Read user input from mouse/trackpad	-
Read user input from keyboard	-
Convert input into booleans and values	-

Model

AlarmModel

Description: This model keeps track of the state of an alarm and draws the alarm based on its state and location.

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

Responsibilities	Collaborators
Draw alarm texture	GameCanvas

BallModel

Description: This model keeps track of the ball's physical properties, including its body, shape, and fixture, and draws the ball differently based on whether it is in movement.

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

Responsibilities	Collaborators
Get information about force acting on ball to move ball	-
Draw ball texture	GameCanvas

ButtonModel

Description: This model keeps track of the state of a button and draws the button based on its state and location.

Justification: This is a model for the button environmental object.

Responsibilities	Collaborators
Draw button texture	GameCanvas

DoorModel

Description: This model keeps track of the state of a door and draws the door based on its state and location

Justification: This is a model for the door environmental object.

Responsibilities	Collaborators
Draw door texture	GameCanvas

GuardModel

Description: This model keeps track of guard properties, including their patrol paths, starting locations, and sight radius and distance. This model draws the guard at its current location.

Justification: This is a model for the guard enemy.

Responsibilities	Collaborators
Get information about guard movement to move guard	-
Draw guard texture	GameCanvas

View

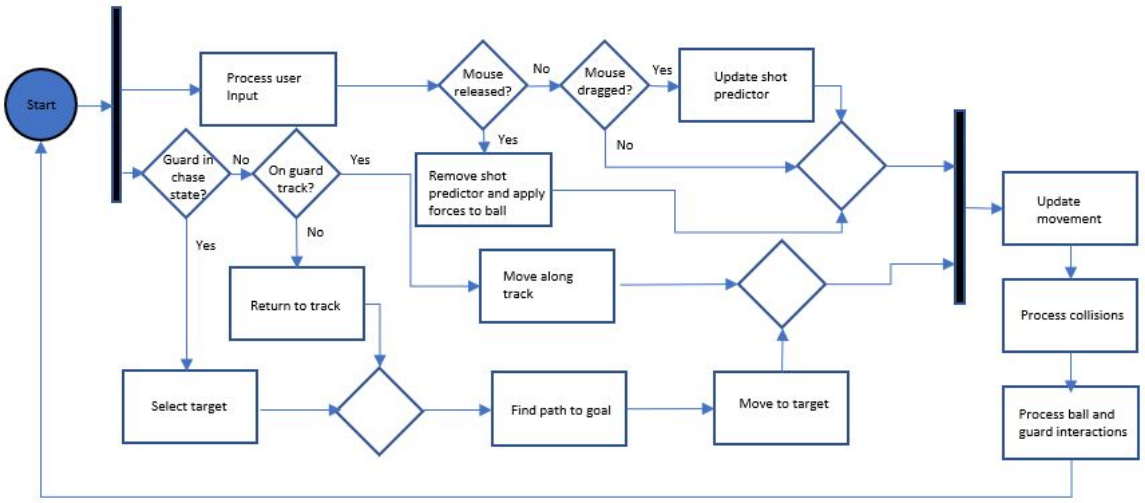
GameCanvas

Description: The game canvas draws textures on the screen at the location provided.

Justification: Game canvas must exist to facilitate the drawing of objects to the screen.

Responsibilities	Collaborators
Draw textures	-

Activity Diagram



Data Representation Model

Saved Game File

Overview

The game will store saved data in a JSON file. This JSON will contain which levels the player has completed, as well as the number of shots taken for those levels.

Example

```
{"levels":[{"number" : 1,
"bestshot" : 1,
},{"number" : 2,
"bestshot" : 2,
},{"number" : 3,
"bestshot" : 4,
}}
```

Level File

Overview

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

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

par: This section of the JSON will be an integer denoting the par of the level for a super pass.

ball_start: This section of the JSON will be a list of length 2 of screen coordinate locations of where the ball should be when the level is initialized. The first element in the list is the x coordinate and the second element in the list is the y coordinate.

ball_goal: This section of the JSON will be a list of length 2 of screen coordinate locations of where the ball should be to reach the goal state. The first element in the list is the x coordinate and the second element in the list is the y coordinate.

walls: This section of the JSON will be a list of arrays of screen coordinate locations of where walls should be placed. In each wall array the even indexes are the x coordinates and the odd indexes are y coordinates. If the coordinate in index i is an x coordinate it goes with the y coordinate in index i + 1. If the coordinate in index i is a y coordinate it goes with the x coordinate in index i - 1.

guard_positions: This section of the JSON will be a list of lists of length 2 of screen coordinate locations of where guards should be placed. The first element in the list is the x coordinate and the second element in the list is the y coordinate.

guard_patrols: This section of the JSON will be a list of arrays 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]. In each guard_patrol array the even indexes are the x coordinates and the odd indexes are y coordinates. If the coordinate in index i is an x coordinate it goes with the y coordinate in index i + 1. If the coordinate in index i is a y coordinate it goes with the x coordinate in index i - 1.

alarms: This section of the JSON will be a list of lists of length 2 of screen coordinate locations of where alarms should be placed. The first element in the list is the x coordinate and the second element in the list is the y coordinate.

doors: This section of the JSON will be a list of arrays of door information. The length of this list will be the same as the length of the list in the buttons section. doors[i] corresponds to the doors associated with the button in buttons[i]. The array representing a specific door has length 5. The first two values are integers representing the x and y coordinate of the door. The third value is an integer representing the ID of the door. The fourth value is a boolean representing whether the door is timed. The last value is an integer representing the number of shots to reset the door.

buttons: This section of the JSON will be a list of arrays of button information. The length of this list will be the same as the length of the list in the door section. buttons[i] corresponds to the button associated with the door in doors[i]. The array representing a specific button has length 5. The first two values are integers representing the x and y coordinate of the button. The third value is an integer representing the ID of the button. The fourth value is a boolean representing whether the button is timed. The last value is an integer representing the number of shots to reset the button.

Example


```
{"level": 1,
 "par": 12,
 "ball_start": [2.375,2.42],
 "ball_goal": [19,16],
 "walls": [
   [0.0, 0.0, 32.0, 0.0, 32.0, 18.0, 0.0, 18.0, 0.0, 17.0, 31.0,
17.0, 31.0, 1.0, 1.0, 1.0, 1.0, 17.0, 0.0, 17.0],
   [14.6875, 11.0625, 14.6875, 16.15625, 17.25, 16.15625, 17.25,
11.0625, 14.6875, 11.0625],
   [14.6875, 7.0625, 14.6875, 12.15625, 17.25, 12.15625, 17.25,
7.0625, 14.6875, 7.0625],
   [20.8875, 11.0625, 20.8875, 16.15625, 23.45, 16.15625, 23.45,
11.0625, 20.8875, 11.0625],
   [20.8875, 7.0625, 20.8875, 12.15625, 23.45, 12.15625, 23.45,
7.0625, 20.8875, 7.0625]
 ],
 "guard_positions": [
   [5.181827,6.345055],
```

```
    [1.59375,9.5]
  ],
  "guard_patrols": [
    [5.181827, 6.345055, 9.34375, 6.34375],
    [1.59375, 9.5, 5.71875, 9.4375]
  ],
  "alarms": [
    [8.78125,3.5625]
  ],
  "doors": [
    [19.05,7.97,4,false,12],
    [19.05,9.97,3,false,12],
    [19.05,11.97,2,false,12],
    [19.05,13.97,1,false,12]
  ],
  "buttons": [
    [25.375,9.42,4,false,12],
    [25.375,3.42,3,false,12],
    [25.375,5.42,2,false,12],
    [25.375,7.42,1,false,12]
  ]

}
```