Dylan LoPresti, Mohamed Bakr, Ryan Mosenkis, Cole Hausman

CSCI 205 Final Project Spring 2021

Professor Dancy - 1:50 p.m. section

Design Manual

Our game is an imitation of the classic Google Dinosaur Game, created utilizing the JavaFX library. Our class structure is set up in the MVC format as well as a separate Controller class for the main menu of the game. The game allows for the user to choose from a variety of images to play as the character in the game, as well as watching an AI play the game instead. The AI functions mainly within the GameController and GameModel classes, utilizing prediction to avoid obstacles and collect food. The game runs until the player/AI and an obstacle collide, at which point the game will terminate. As the game progresses the score of the player will constantly increase, gaining bonuses for each food object eaten. The obstacles gain speed as the score and subsequently the game goes on.

User Stories:
As a player I want to be able to jump over obstacles with simulated physics - completed
As a player I want to be able to interact with objects within the game - completed
As a player I want sidescrolling obstacles to spawn within the game to make it more challenging - completed
As a player, I would like the "default character" to be a bison - completed
As a player, I would like to choose from multiple characters to play with - completed
As a player, I would like a way to tell how far I got in a game - completed
As a player, I would like to be able to have at least two different ways of moving the character - completed
As a player, I would like the option to play the game myself, or see how the computer does - completed
As a player, I would like the character to be able to eat food to earn extra points - completed
As a player, I would like there to be a help menu to learn how the game works - completed

OOD Design:

CRC Cards:
Our design mainly utilizes JavaFX library to create a th Game. One of the best design strategies to conquer such a design is to use the Model-View-Control (MVC) approach. The model in the MVC deals with the abstract logic and data in our system (back-end) . The View is essentially the rendering of the graphical user interface (GUI) and everything displayed on the screen (front-end). Finally, the Control is the underlying logic that connects the model and the view together tying everything together yielding a working system.

GameModel	
• Handle player jump • Handle obstacle movement • Handle food movement • Handle collisions • Handle calculations of AI player	• GameView • GameMain • GameControl

GameControl	
• Handle user key inputs • Make decisions based on observations made by AI • Determine game states based on collisions	• GameView • GameModel • GameMain

GameView	
• Store player object • Store player Image to display • Store obstacle object • Store obstacle food to display • Store food object • Store food image to display • Handle music • Displays all stored objects and images to the main stage	• GameControl • GameModel • GameMain

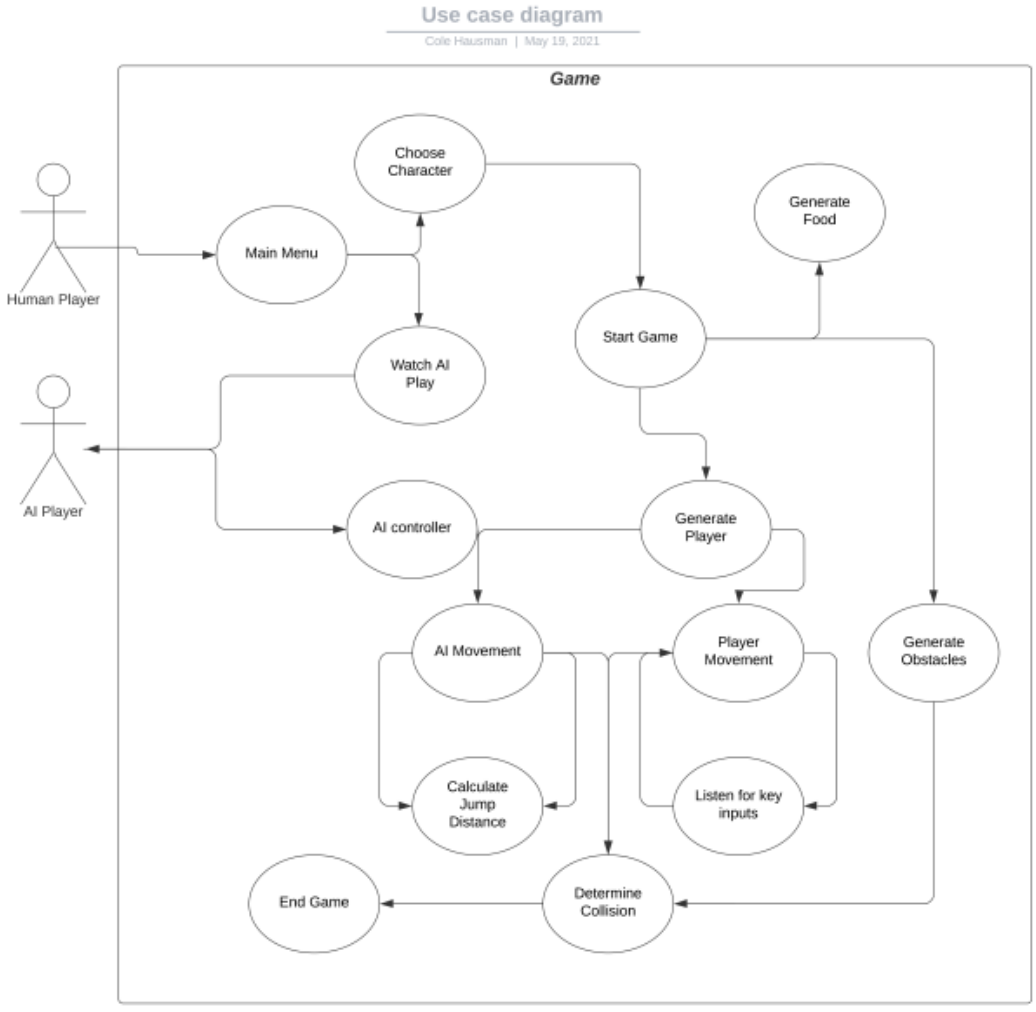
MenuController	
• Allows user to choose character image • Allows user to choose between playing and watching AI play • Shows help instructions • Starts the game	• GameMain • GameView • GameControl

There are four main classes which run our game and interact with one another. The GameModel contains most of the functions and delivers their functionality to GameView, GameControl and also the GameMain. GameControl utilizes the functions in GameModel in order to determine game states with logic. These two classes work together to do much of the logic based aspects of the game and are also responsible for the brain of the AI. The GameView class is what stores all of the nodes and images and even music. This class allows for everything to be displayed to the main stage so that the user can interact with the game in a meaningful way. Finally, the MenuController is what allows the menu to interact with and change

game states such as starting the game. This class works with the three aforementioned classes as well as GameMain which is not listed simply because of its lack of importance in the grand scheme of the game's MVC. The MenuController class also stores the help instructions, allows for character selection, and is what toggles the AI to be on or off.

Use Case Diagram:

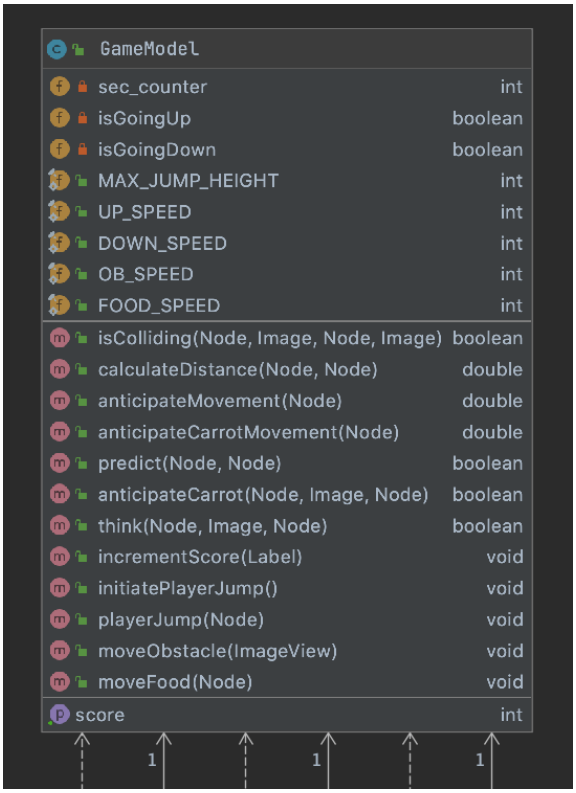
This Use Case Diagram how our game is intended to be run as well as how the user is expected to interact with the game. The player first determines in the main menu whether to play the game themselves or to let the AI play the game for them. If they play themselves, then they choose a character and the game begins. If they choose the AI, then the AI controller is activated. The game is started either way and begins generating the food, the obstacles, and displays the player. If the AI is on then the movement is calculated based on object positions, if the player is playing then the game listens for their key inputs to determine if the player is jumping or not. Finally, if a player, AI or not, collides with any obstacle, the game ends.



GameModel:

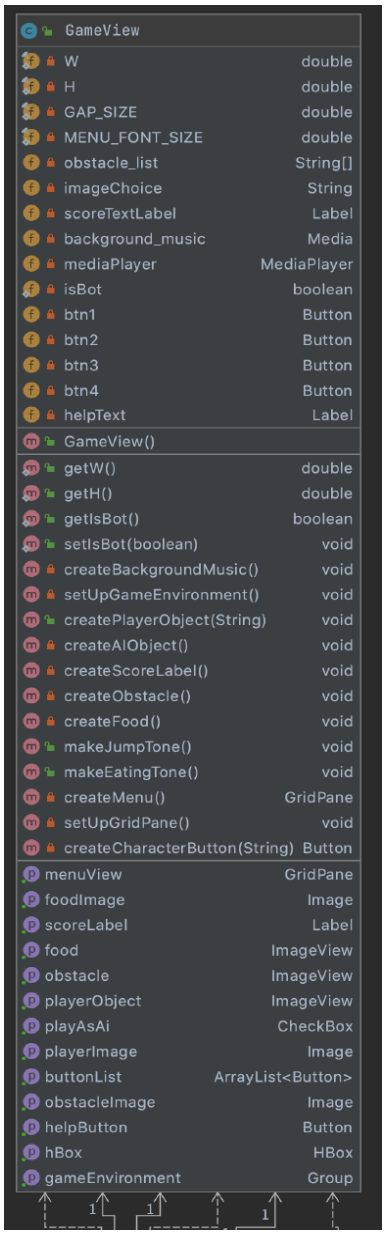
We start out OOD design at the GameModel class,this class has the underlying logic and data of the game. The GameModel does not depend on any other class, but other classes definitely depend on it (just like the UML class diagram here shows).

The GameModel includes important data fields like sec_counter that manages increasing the score, boolean values needed for the jumping, and speeds and heights needed for different objects. The methods in GameModel deal with logic surrounding the game. This includes all the AI functions (calculateDistance, anticipateMovment, anticipateCarrotMovment, predict, think). It also manages incrementing the score through incrementScore. It manges moving objects either the food or the obstacles, from one side of the screen to the other. Finally, it deals with the underlying logic for making an object jump to a certain height.



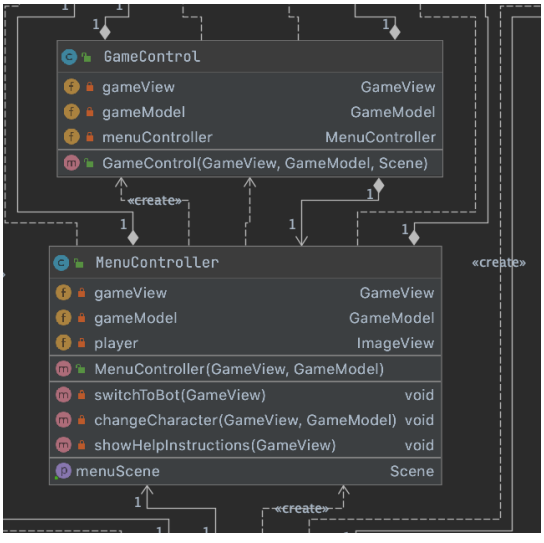
GameView:

Next step in our OOD design is the GameView class. The GameView class deals with all the GUI needed for the game including that of the menu. Views typically depend on the Model, but in our case, we did not need that. So, the GameView class also does not depend on any other class, but again other classes depend on it (Again, you can see that in the UML class diagram below)



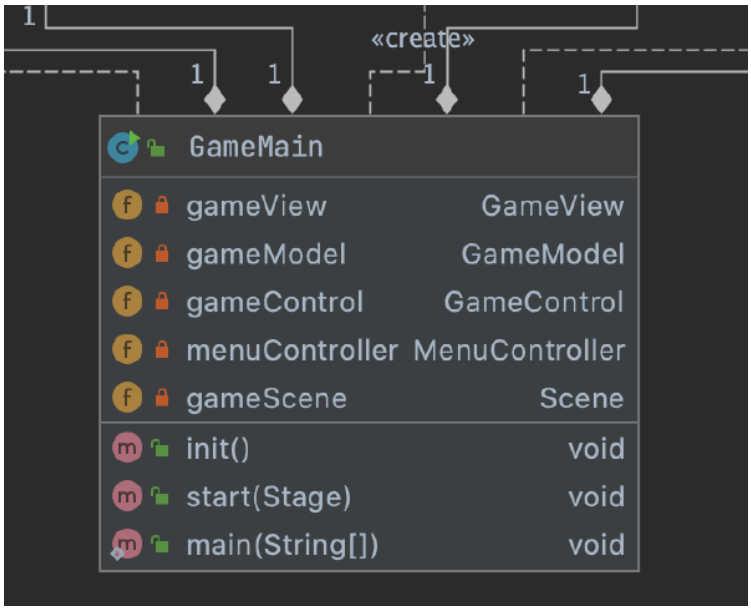
The GameView includes all sorts of fields related to the GUI. It uses double values like for Width and Height of the scene. It also uses several types of JavaFX objects including, Labels, Media and MediaPlayers, Buttons, Images and ImageViews, HBox, and GridPane. It also uses several other data types that aid the process of creating the GUI like Strings and ArrayLists. Moving on to the fields, the main GameView method calls in all other helper methods in the class. It starts by creating the main menu with all the character buttons, checkBox and help button. It then moves on to create the obstacle, food, scoreLabel and the player whether an AI or an actual player. Finally, it creates the background media and sets up all of this in the gameEnvironment. So when GameView is instantiated, it essentially creates the whole GUI of both the menu and the gameEnvironment. Which of these to actually get to show on the screen is controlled in another class (MenuController) as we will see later on.

MenuController and GameControl:
These two classes together represent the underlying logic that connect the view and the model together. MenuController deals with controlling the menu GUI while GameControl deals with controlling the gameEnvironment.



We can see from the UML class diagram above that the MenuController actually depends on both GameView and GameModel, which is what a controller usually needs. It also creates the GameControl class inside of it (hence the create usage relationship between them).So, when the MenuController is instantiated, it first uses the GameView to create the MenuScene that shows up in the beginning of the game. It then deals with all the functionality of the menu. For example, clicking on the help button would display the label that shows the instruction on how to play the game and clicking anywhere will take you back to the menu. Checking the AI checkBox will let you use the AI when starting the game. Finally, choosing one of the characters will let you play with that specific character and then create the GameControl passing the menuSciene to it. When the GameControl is created, it changes the scene root to the gameEnvironment from the view thus moving to the actual game. Here the GameControl either lets you jump if you choose to play the actual game. It also deals with incrementing the score as time goes by. It moves the obstacles, the food, increments the score when you eat the food. It also has the underlying logic that lets the AI jump the obstacles on its own. It also adds a tone to jumping and eating food. Finally, it deals with ending the game when you collide with an obstacle.
Note that when the game ends we do not move back to the menu since the underlying mechanism uses animationTimer which causes a lot of problems and errors when trying to move back to the menu, that is why we end the game completely as you hit an obstcale.

GameMain:
Final step in our OOD design is the GameMain. This is the main class where the Java Application is actually extended. It depends on all other classes. It also creates the GameView, MenuConroller, GameModel (as you can see in the UML class diagram below). Once the MenuController returns the main scene, GameMain is where it is staged and shown. All other functionality trickles down from MenuController, GameView and GameModel as explained.



JUnit Tests:
Since all of our OOD design is based on GUI that runs using JavaFX, JUnit testing is almost impossible. We tried to create the most basic JUnit tests for the GameModel methods like implement score and collide which is the most abstract class that we have, but nothing would make the test pass even though the logic looked perfectly fine. The Testing part of this design was mainly looking at the actual game and seeing if some functionality works or not.