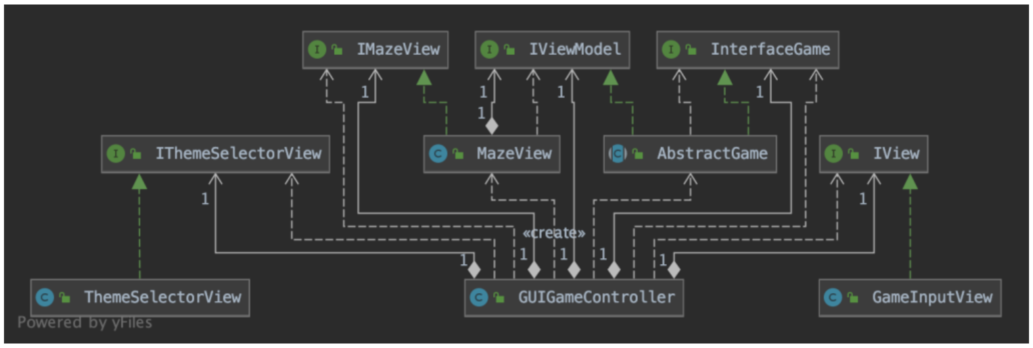
HW6 HTW View

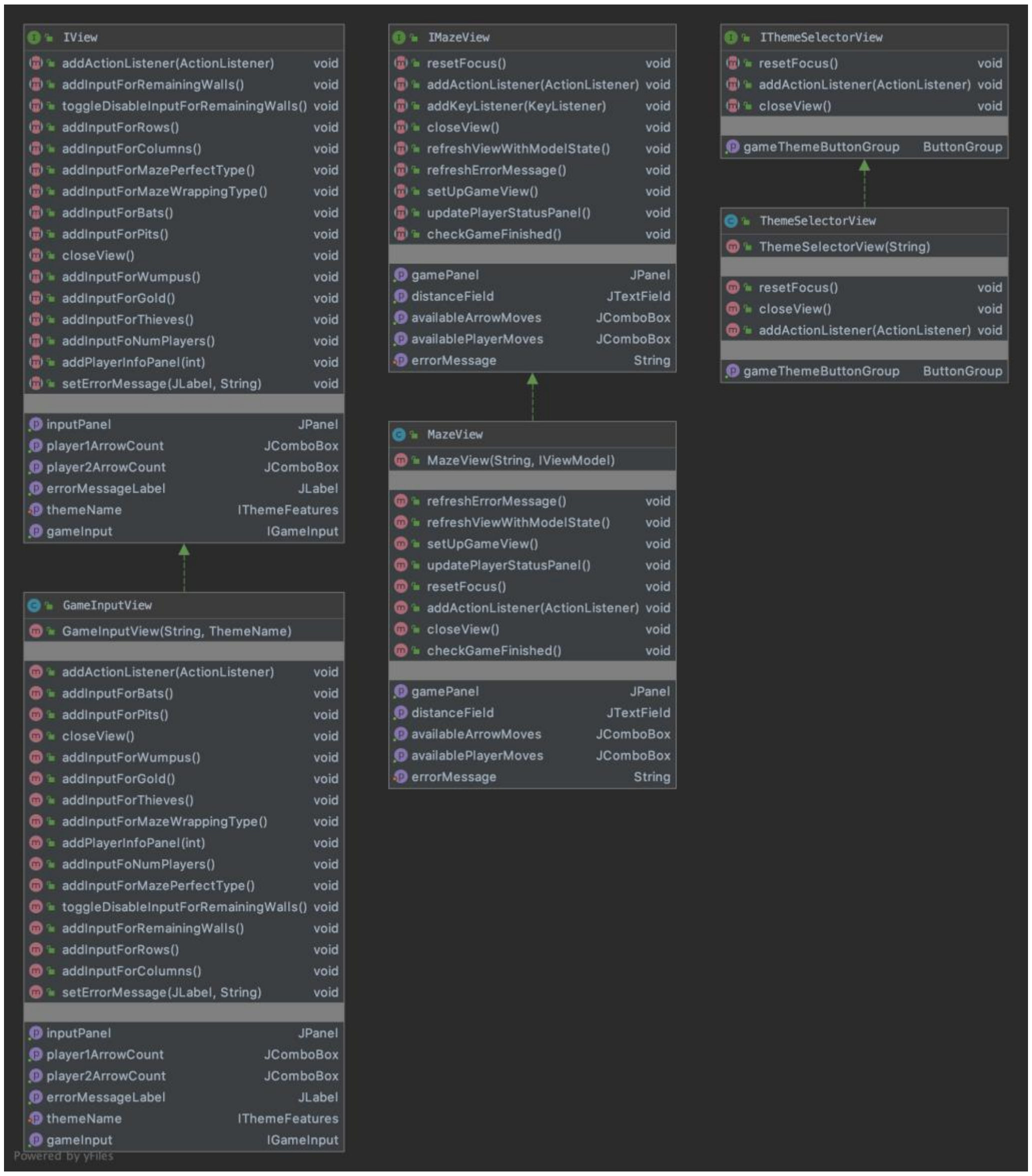
1. UML Diagram

MVC Architecture

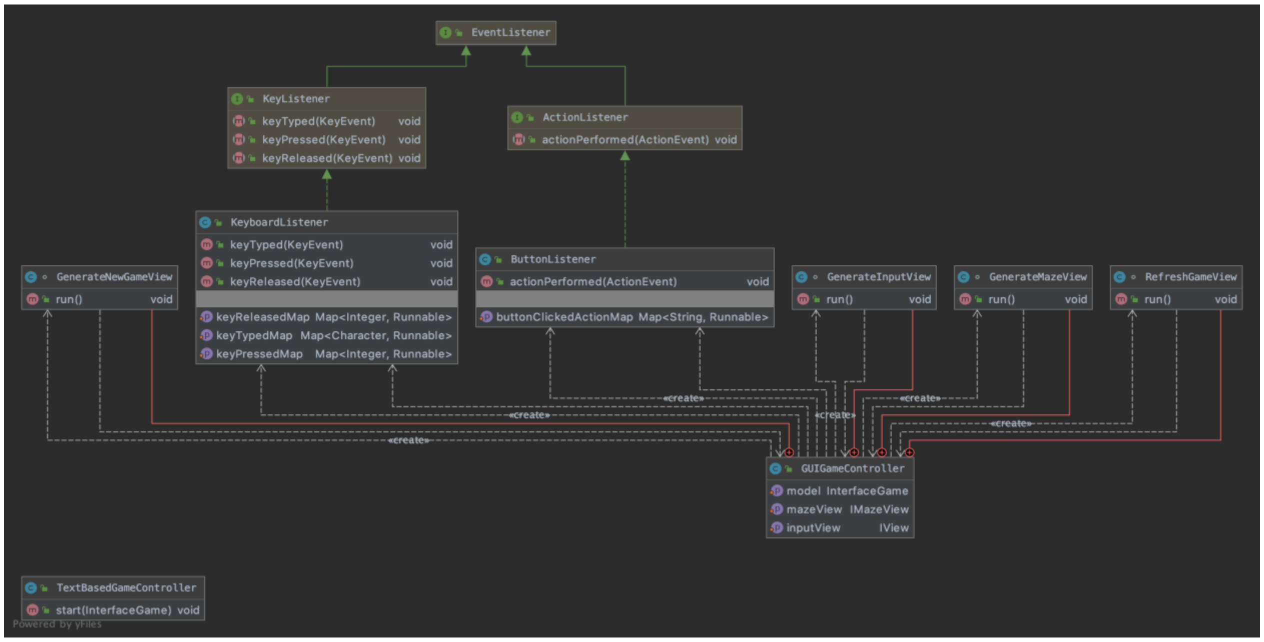
Model: InterfaceGame/AbstractGame
View: IThemeSelectorView/IMazeView/IView
Controller: GUIGameController



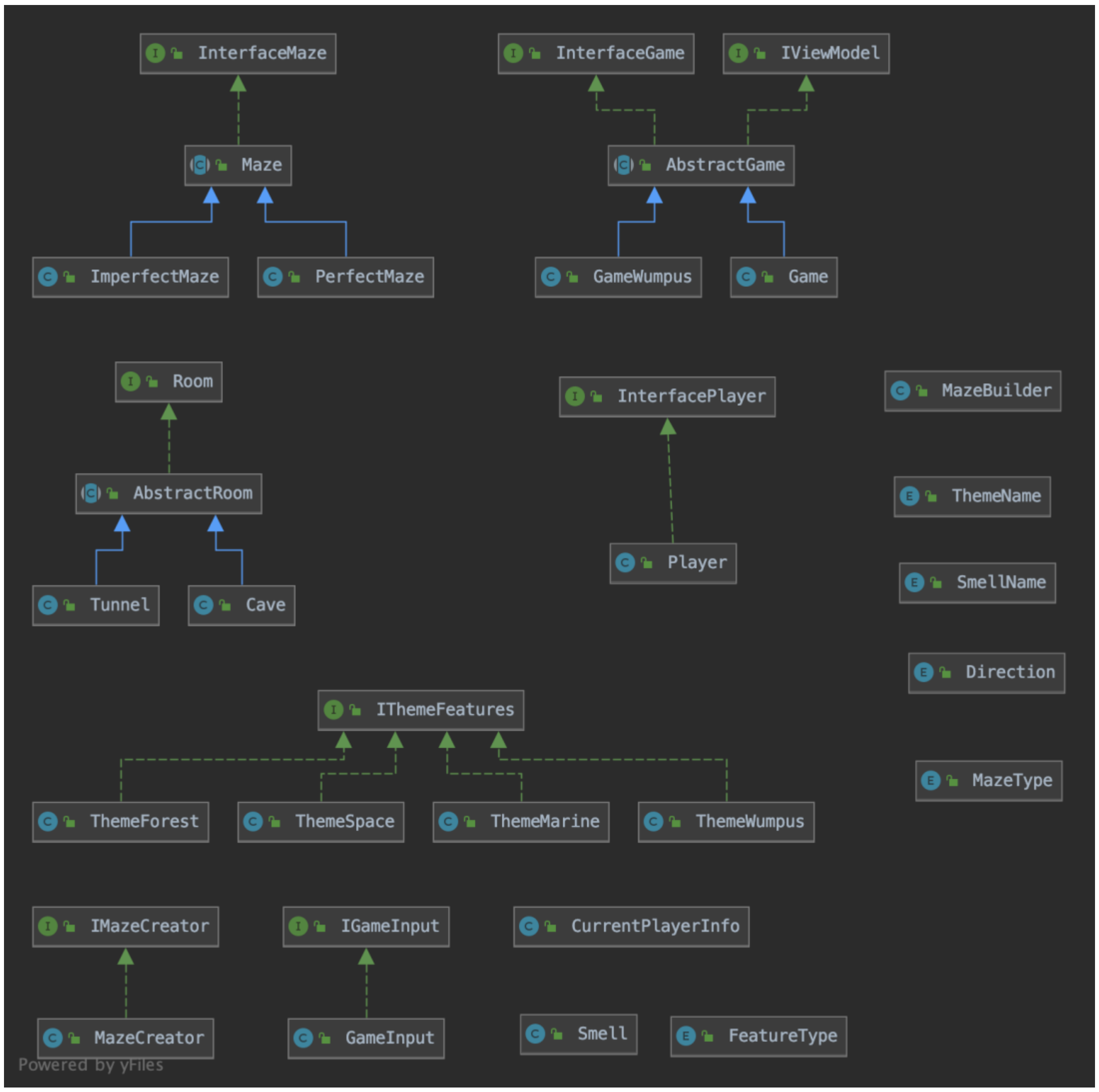
View Interfaces and view classes


IView
addActionListener(ActionListener) void
addInputForRemainingWalls() void
toggleDisableInputForRemainingWalls() void
addInputForRows() void
addInputForColumns() void
addInputForMazePerfectType() void
addInputForMazeWrappingType() void
addInputForBats() void
addInputForPits() void
closeView() void
addInputForWumpus() void
addInputForGold() void
addInputForThieves() void
addInputFoNumPlayers() void
addPlayerInfoPanel(int) void
setErrorMessage(JLabel, String) void
inputPanel JPanel
player1ArrowCount JComboBox
player2ArrowCount JComboBox
errorMessageLabel JLabel
themeName IThemeFeatures
gameInput IGameInput
GameInputView
GameInputView(String, ThemeName)
addActionListener(ActionListener) void
addInputForBats() void
addInputForPits() void
closeView() void
addInputForWumpus() void
addInputForGold() void
addInputForThieves() void
addInputForMazeWrappingType() void
addPlayerInfoPanel(int) void
addInputFoNumPlayers() void
addInputForMazePerfectType() void
toggleDisableInputForRemainingWalls() void
addInputForRemainingWalls() void
addInputForRows() void
addInputForColumns() void
setErrorMessage(JLabel, String) void
inputPanel JPanel
player1ArrowCount JComboBox
player2ArrowCount JComboBox
errorMessageLabel JLabel
themeName IThemeFeatures
gameInput IGameInput
IMazeView
resetFocus() void
addActionListener(ActionListener) void
addKeyListener(KeyListener) void
closeView() void
refreshViewWithModelState() void
refreshErrorMessage() void
setUpGameView() void
updatePlayerStatusPanel() void
checkGameFinished() void
gamePanel JPanel
distanceField JTextField
availableArrowMoves JComboBox
availablePlayerMoves JComboBox
errorMessage String
MazeView
MazeView(String, IViewModel)
refreshErrorMessage() void
refreshViewWithModelState() void
setUpGameView() void
updatePlayerStatusPanel() void
resetFocus() void
addActionListener(ActionListener) void
closeView() void
checkGameFinished() void
gamePanel JPanel
distanceField JTextField
availableArrowMoves JComboBox
availablePlayerMoves JComboBox
errorMessage String
IThemeSelectorView
resetFocus() void
addActionListener(ActionListener) void
closeView() void
gameThemeButtonGroup ButtonGroup
ThemeSelectorView
ThemeSelectorView(String)
resetFocus() void
closeView() void
addActionListener(ActionListener) void
gameThemeButtonGroup ButtonGroup
Powered by yFiles

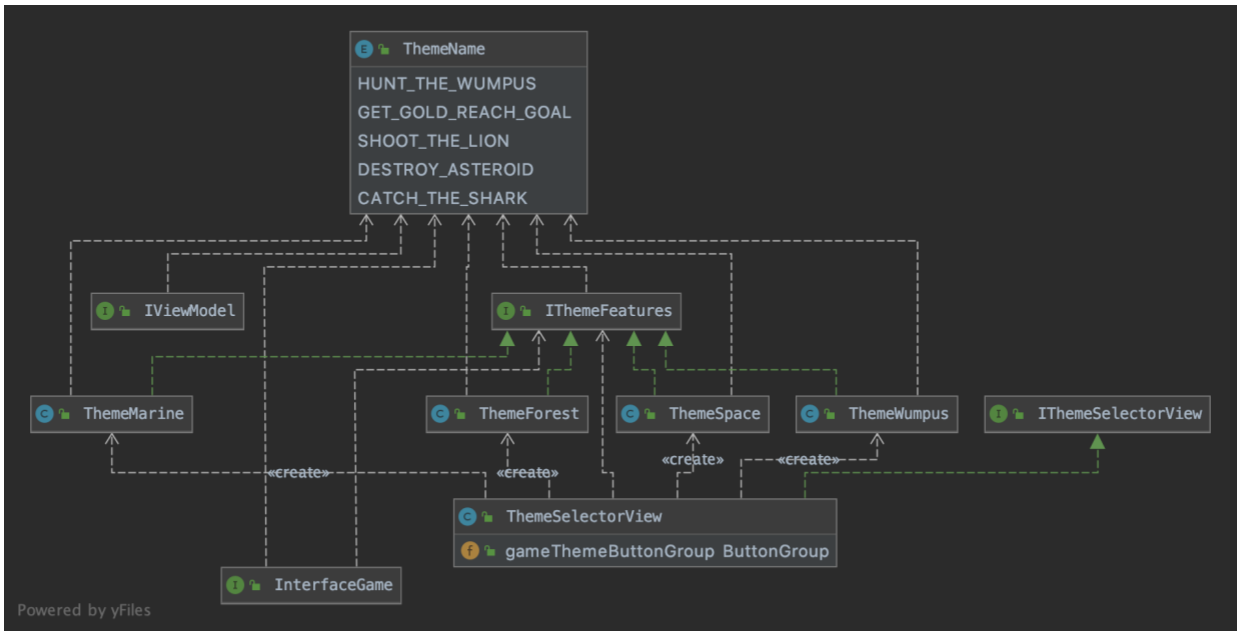
Controller:



Model:



Themes:


ThemeName
HUNT_THE_WUMPUS
GET_GOLD_REACH_GOAL
SHOOT_THE_LION
DESTROY_ASTEROID
CATCH_THE_SHARK
IViewModel
IThemeFeatures
ThemeMarine
ThemeForest
ThemeSpace
ThemeWumpus
IThemeSelectorView
«create»
«create»
«create»
«create»
ThemeSelectorView
gameThemeButtonGroup ButtonGroup
InterfaceGame
Powered by yFiles

2. Design

1. Added different packages for models, views and controllers for ease of code reading.
2. Added a IView, IMazeView, IThemeSelectorView interfaces for all 3 views - ThemeSelectorView, GameInputView and MazeView. Created different type of interfaces for different views because there were extremely few common methods. It was a fair tradeoff rather than having numerous methods throw UnSupported exception.
3. Added a GUIGameController for controlling GUI based game by being the intermediary between Game models and views
4. features for different themes to dynamically generate the input form, so created a IThemeFeature interface. Implemented ITheme interface and classes ThemeWumpus, ThemeMarine, ThemeSpace and ThemeForest to return features specific to individual games.
5. view to convey all the Maze generation inputs to the model, rather than conveying it for every feature, wrapped all the data in a object of GameInput. Created IGameInput interface and implemented the same in GameInput class.
6. Starting position is now selected randomly.
7. Creating an interface IMazeCreator and corresponding class MazeCreator which takes in all the input provided by the user and verifies for any errors. After error checking, it would create an instance of a maze and return the maze to the controller which will be an input to the Game model.
8. In HW5, my player's status like alive/dead and won/lost was stores in my Game class. Moved that to the Player class. Also added getters and setters for the same.
9. Added a new constructor GameWumpus(InterfaceMaze maze, List<Player> players, int randomSeed) to tackle the feature of multi-players.Correspondingly made a similar constructor in super() ie AbstractGame which assigns the multi-players to the maze.
10. IViewModel with read-only methods about the model which the view can pull as desired.
11. Implemented Button and Action listeners to take care of all activity.
12. Implemented corresponding methods in controller across each listener.

3. Test Plan

Testing the view:

1 player mode:

UI view:

1. Active player turn is correctly displayed. Game starts with player1.
2. Active player icon is correctly displayed. Game starts with player1.
3. Count of weapons with player is correctly displayed
4. Location of player is correctly displayed
5. Player starts at a location not occupied by Wumpus/bat/pit.
6. The maze is invisible at the start.
7. Grid becomes visible when player enters a room.
8. Player loses upon entering grid with Wumpus.
9. Player loses upon entering grid with Pit.
10. Player loses when arrows are over.
11. Player is transported upon entering room with Bat.
12. Player wins by shooting the Wumpus
13. Breeze is displayed in cells adjacent to Wumpus.
14. Stench is displayed in cells adjacent to Pit.

Button clicks

1. Player can move in all valid directions on clicking **Move button**. Player does not move upon an incorrect direction input.
2. Player can shoot in all available directions for any distance on clicking **Shoot button**. Arrow count reduces upon one shoot.

3. Distance while shooting if negative is handled properly.
4. Pressing the **Exit button** ends the game
5. Pressing the **Restart button** ends the game
6. Pressing the **New game button** fetches the Menu screen

Keyboard presses

1. Player can move in all valid directions using **arrow keys**. Player does not move upon an incorrect direction input.

2 player mode:

UI view:

1. Both the players are visible at the start of the game. Active player icon is correctly displayed. Game starts with player1.
2. Count of weapons with player1 and player2 is correctly displayed
3. Location of player1 and player2 is correctly displayed
4. Player1 and player2 starts at a location not occupied by Wumpus/bat/pit.
5. Grid becomes visible when either of the player enters a room.
6. Active player loses upon entering grid with Wumpus. The other player still continues with the game.
7. Active player loses upon entering grid with Pit. The other player still continues with the game.
8. Active player loses when his arrows are over. The other player still continues with the game.
9. Active player is transported upon entering room with Bat.
10. Active player wins by shooting the Wumpus. The other player loses and game ends.
11. Breeze is displayed in cells adjacent to Wumpus.
12. Stench is displayed in cells adjacent to Pit.

Button clicks

7. Active player can move in all valid directions on clicking **Move button**. Player does not move upon an incorrect direction input.
8. Active player can shoot in all available directions for any distance on clicking **Shoot button**. Arrow count reduces upon one shoot.
9. Distance while shooting if negative is handled properly.
10. Pressing the **Exit button** ends the game
11. Pressing the **Restart button** ends the game
12. Pressing the **New game button** fetches the Menu screen

Keyboard presses

2. Active player can move in all valid directions using **arrow keys**. Player does not move upon an incorrect direction input.

Testing the controller:

Controller is tested by creating a mock model and simulating different commands. Simulate the following clicks during testing:

Button clicks /keyboard presses

1. **Shoot button** shoots an arrow. Arrow count reduces upon one shoot.
2. **Move button** moves the player in selected direction
3. Pressing the **Exit button** ends the game
4. Pressing the **Restart button** ends the game
5. Pressing the **New game button** fetches the Menu screen
6. **Up arrow key** moves the player Up if path exists

7. **Down arrow key** moves the player Down if path exists
8. **Left arrow key** moves the player Left if path exists
9. **Right arrow key** moves the player Right if path exists

Testing the model:

Game initialization

Upon the inputs received from the user, game model gets correctly created on the following factors:

a. Number of rows
b. Number of columns
c. % of bats
d. % of pits
e. 1 Wumpus
f. Perfect/Imperfect
g. Wrapping/Non-wrapping
h. Number of players

Movement

a. Player is able to move in all valid directions
b. Incorrect direction is handled and does not move the player
c. Active player loses upon entering grid with Wumpus. The other player still continues with the game.
d. Active player loses upon entering grid with Pit. The other player still continues with the game.
e. Active player is transported upon entering room with Bat.

Shooting

a. Player is able to shoot in all valid directions and positive distances
b. Incorrect direction is handled and does not shoot the arrow
c. Negative/fractional distance is handled and does not shoot the arrow
d. Active player wins upon shooting in grid with Wumpus. The other player loses.
e. Active player loses when his arrows are over. The other player still continues with the game.
f. Wumpus is not killed if arrow lands into a cave with no Wumpus or is stopped by a wall

Player status

a. Getter method for getting location and arrow count of active player
b. Getter method for getting error message upon incorrect input
c. Getter method for getting notification message upon movement/shoot