Name of Project: Pacman
Team Members: Rohan Suri, Max Macaluso, Sahib Bajwa

Final State of System

The final state of the system has 3 different states loosely based on the interactions indicated in our use case diagrams from project 4. The first state will be the Start/Main Menu where the player will have the opportunity to start the game by clicking a button with their mouse click. The second state will be actual gameplay where the player will be indicated as a pacman player on the board along with 4 ghosts (enemies) and coins. Coins are statically defined before the game starts and enemies are dynamically moving each with different behaviors on how they will pursue the pacman player. The player will be able to collect or land on coins on the board which will allow them to increase their score and players will also have lives that will be subtracted each time they collide with any one of the ghosts (enemies). Movement by the player will be done using keyboard input of the 4 arrow keys. The 3rd and final state will be the game over screen where the player has lost all of their lives and their score is shown on the board. Overall these 3 states encompass most of the requirements we indicated in our Project 4 iteration, but something we were not able to implement was the frightened mode of ghosts (enemies) where the pacman player can pick up an energizer which changes the enemies color to blue, behavior to frightened, and gives the pacman player an opportunity to collide with the ghosts to earn more points rather than losing a life. We also were not able to implement multiplayer gameplay due to time constraints. Other than that, we were able to meet all of the other requirements indicated in project 4.

Final Class Diagram and Comparison Statement

We didn’t have many changes from our original project 4 design to our final project 6 design, however, we did divide some class functionality to make our code cleaner. Firstly, we moved all non-GUI components of UI into a separate class called Driver.py and changed the name UI to UIClass. Next, we decided to optimize/change our search algorithm for the enemy ghost targeting the player from BreadthFirstSearch(start, target) to AStarSearch(start, target). Thirdly, we decided to change the GameAnnouncer Class to Observer Class and removed the relevant interfaces in the previous UML class diagram because python has very limited support for interfaces making this a challenge of a non-OOP language. The first UML Diagram below labelled Project 4 was our initial UML Diagram and the second UML Diagram is our finalized UML Diagram.

Third Party Vs. Original Code

Most of our code was original, however, we did use some sources to aid our design pattern development, search algorithm decisions and utilization of the pygame library. The sources are indicated below.

We used the first source to implement the singleton pattern in python since we only learned how to do so in Java in this class. We used the second source to brainstorm the classes we would need. We ultimately decided we would not use parsers and the standard “game” library as we found pygame to be better suited for our experience and scope of the project. The third source gave us the idea to map our board out in a text file with locations of player, enemies, and coins to use alongside the pygame library to map pixels on a pacman board image to the symbols specified in a text file. We also saw that they used BFS for Ghosts to seek players, but we ultimately decided to use an A* Pathfinding Algorithm to optimize enemy performance and make the game even more difficult. The fourth source helped us understand how to take in keyboard inputs via the pygame library.

- https://python-patterns.guide/gang-of-four/singleton/
- https://www.cse.wustl.edu/~garnett/cse511a/code/project2/pacman_py.html
- https://github.com/a-plus-coding/pacman-with-python
- https://magpi.raspberrypi.org/articles/code-pac-man-in-python

Additionally, we used the sources below, but less frequently:

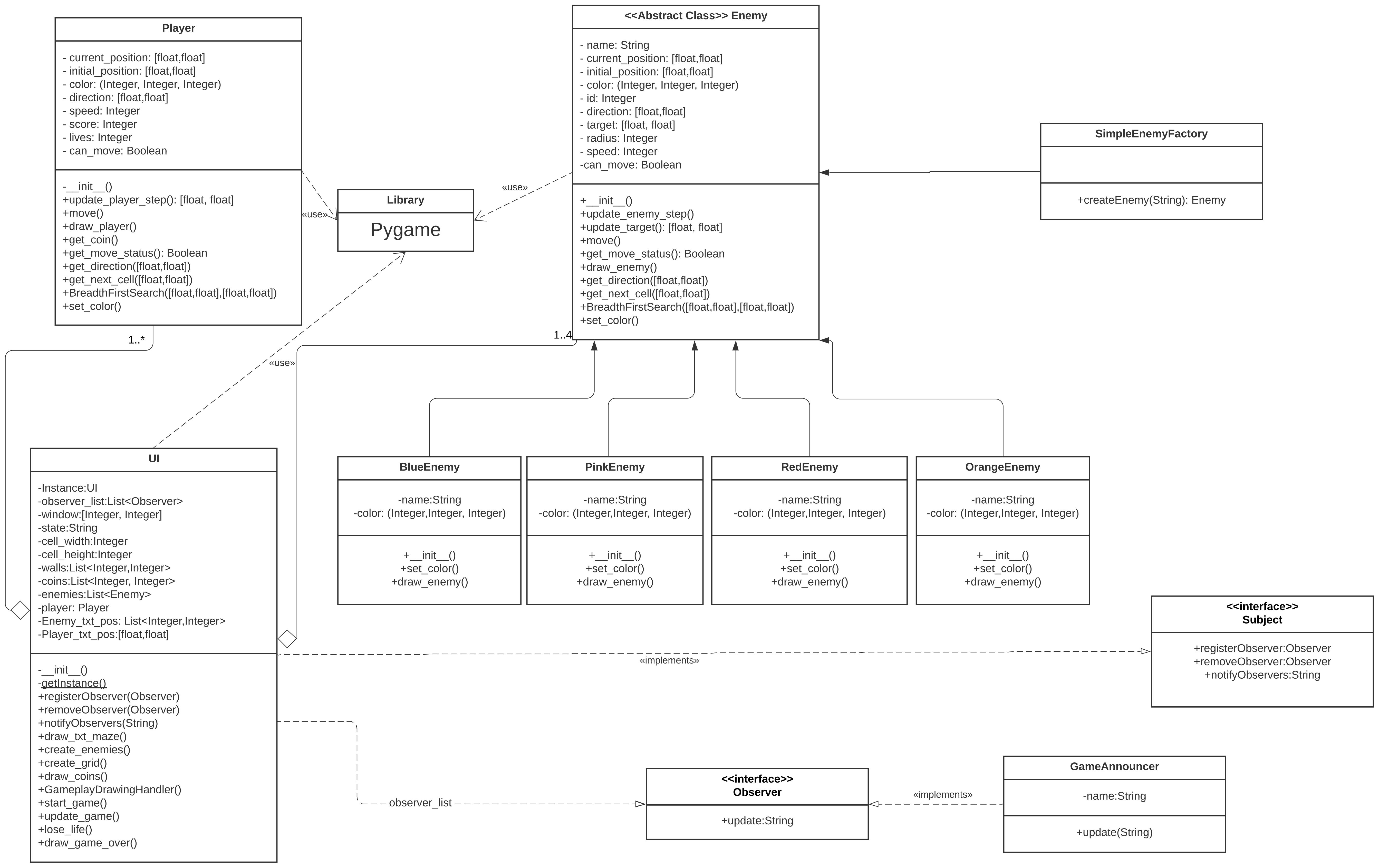
1. https://www.youtube.com/watch?v=juSH7hmYUGA (Timer Tutorial)
2. https://docs.python.org/3/library/enum.html (Enum Tutorial)
3. https://pypi.org/project/pathfinding/ (Pathfinding Library/Tutorial)
4. http://zetcode.com/javagames/pacman/ (Referenced Code)
5. https://www.geeksforgeeks.org/factory-method-python-design-patterns/ (Factory Pattern Tutorial)
6. https://refactoring.guru/design-patterns/observer/python/example (Observer Pattern Tutorial)

Statement on the OOAD process for your overall Semester Project

In our team's design process, we encountered many different situations, some of which were good while others could benefit from improvement and were likely induced by the remote nature of our work this year. One such area is (1) communication: around Thanksgiving our team, as a whole, stopped communicating several days and ultimately led to us having to push back our submission date to 12/07/20. However, it is important to note that our communication was not all bad. Overall, our team did a great job staying in touch with each other, would meet most weeks, and actively maintained a group chat. Another design process element that is prevalent in industry as well with larger projects is (2) merge conflicts. In an attempt to conflict this problem, we all set schedules with tasks that we would complete in a certain time period in hopes that the next person would ideally be able to pick up where the other person left off. If we had more time, I would have liked to formalize a better system and/or plan to resolve this issue, although I experienced developers even encounter this issue. Lastly, the final design process issue our team encountered was related to (1) coding style. I see now why many companies in industry produce their own programming style guidelines and enforce them so strictly. It not only improves readability, but also makes the code itself less prone to bugs (erros) because everyone is on the same page.

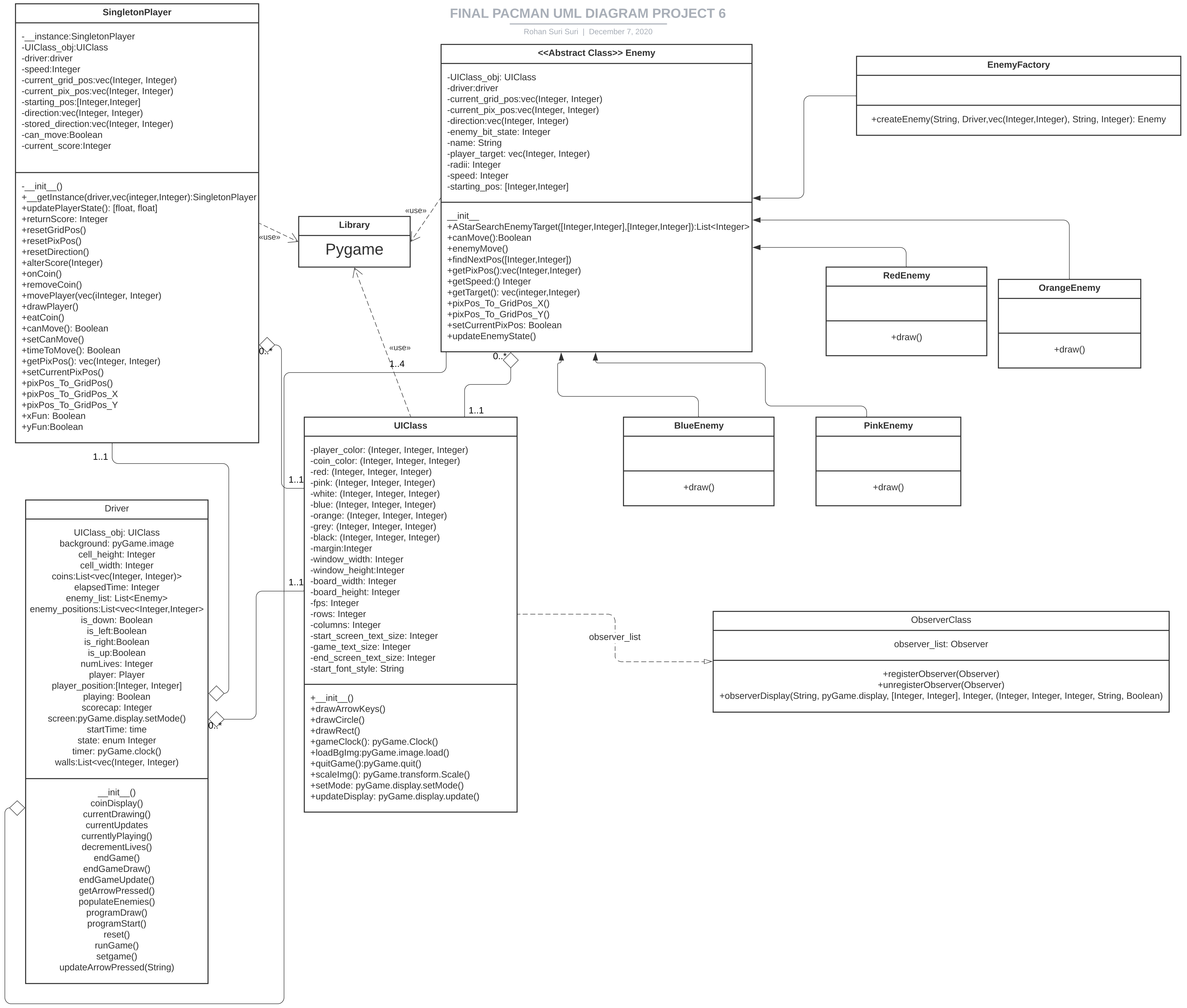
Pacman UML Class Diagram P4

Rohan Suri Suri | December 7, 2020



FINAL PACMAN UML DIAGRAM PROJECT 6

Rohan Suri Suri | December 7, 2020


SingletonPlayer
-__instance:SingletonPlayer
-UIClass_obj:UIClass
-driver:driver
-speed:Integer
-current_grid_pos:vec(Integer, Integer)
-current_pix_pos:vec(Integer, Integer)
-starting_pos:[Integer,Integer]
-direction:vec(Integer, Integer)
-stored_direction:vec(Integer, Integer)
-can_move:Boolean
-current_score:Integer
-__init__()
+__getInstance(driver,vec(integer,Integer):SingletonPlayer
+updatePlayerState(): [float, float]
+returnScore: Integer
+resetGridPos()
+resetPixPos()
+resetDirection()
+alterScore(Integer)
+onCoin()
+removeCoin()
+movePlayer(vec(iInteger, Integer)
+drawPlayer()
+eatCoin()
+canMove(): Boolean
+setCanMove()
+timeToMove(): Boolean
+getPixPos(): vec(Integer, Integer)
+setCurrentPixPos()
+pixPos_To_GridPos()
+pixPos_To_GridPos_X
+pixPos_To_GridPos_Y
+xFun: Boolean
+yFun:Boolean
Library
Pygame
«use»
«use»
«use»
1..4
0..*
1..1
1..1
1..1
0..*
0..*
1..1
<<Abstract Class>> Enemy
-UIClass_obj: UIClass
-driver:driver
-current_grid_pos:vec(Integer, Integer)
-current_pix_pos:vec(Integer, Integer)
-direction:vec(Integer, Integer)
-enemy_bit_state: Integer
-name: String
-player_target: vec(Integer, Integer)
-radii: Integer
-speed: Integer
-starting_pos: [Integer,Integer]
__init__
+AStarSearchEnemyTarget([Integer,Integer],[Integer,Integer]):List<Integer>
+canMove():Boolean
+enemyMove()
+findNextPos([Integer,Integer])
+getPixPos():vec(Integer, Integer)
+getSpeed:() Integer
+getTarget(): vec(integer,Integer)
+pixPos_To_GridPos_X()
+pixPos_To_GridPos_Y()
+setCurrentPixPos: Boolean
+updateEnemyState()
EnemyFactory
+createEnemy(String, Driver,vec(Integer,Integer), String, Integer): Enemy
RedEnemy
+draw()
OrangeEnemy
+draw()
BlueEnemy
+draw()
PinkEnemy
+draw()
UIClass
-player_color: (Integer, Integer, Integer)
-coin_color: (Integer, Integer, Integer)
-red: (Integer, Integer, Integer)
-pink: (Integer, Integer, Integer)
-white: (Integer, Integer, Integer)
-blue: (Integer, Integer, Integer)
-orange: (Integer, Integer, Integer)
-grey: (Integer, Integer, Integer)
-black: (Integer, Integer, Integer)
-margin:Integer
-window_width: Integer
-window_height:Integer
-board_width: Integer
-board_height: Integer
-fps: Integer
-rows: Integer
-columns: Integer
-start_screen_text_size: Integer
-game_text_size: Integer
-end_screen_text_size: Integer
-start_font_style: String
+__init__()
+drawArrowKeys()
+drawCircle()
+drawRect()
+gameClock(): pyGame.Clock()
+loadBgImg:pyGame.image.load()
+quitGame():pyGame.quit()
+scaleImg(): pyGame.transform.Scale()
+setMode: pyGame.display.setMode()
+updateDisplay: pyGame.display.update()
observer_list
ObserverClass
observer_list: Observer
+registerObserver(Observer)
+unregisterObserver(Observer)
+observerDisplay(String, pyGame.display, [Integer, Integer], Integer, (Integer, Integer, Integer, String, Boolean)
Driver
UIClass_obj: UIClass
background: pyGame.image
cell_height: Integer
cell_width: Integer
coins:List<vec(Integer, Integer)>
elapsedTime: Integer
enemy_list: List<Enemy>
enemy_positions:List<vec<Integer,Integer>
is_down: Boolean
is_left:Boolean
is_right:Boolean
is_up:Boolean
numLives: Integer
player: Player
player_position:[Integer, Integer]
playing: Boolean
scorecap: Integer
screen:pyGame.display.setMode()
startTime: time
state: enum Integer
timer: pyGame.clock()
walls:List<vec(Integer, Integer)
__init__()
coinDisplay()
currentDrawing()
currentUpdates
currentlyPlaying()
decrementLives()
endGame()
endGameDraw()
endGameUpdate()
getArrowPressed()
populateEnemies()
programDraw()
programStart()
reset()
runGame()
setgame()
updateArrowPressed(String)

Learn about this template

UML class diagrams map out the structure of a particular system by modeling its classes, attributes, operations, and relationships between objects.

To customize this template:

- *Click on any shape and type the information you would like to include.*
- *Add and arrange class shapes as needed.*
- *Update cardinality.*
  - *Click on a line and navigate to the properties bar to adjust the endpoints.*
  - *Click on a line and hover over the gear icon to add multiplicities.*
  - *Add additional lines by hovering over a shape and clicking the red dot*

UML Class Diagram Tutorials

(Hold Shift + ⌘ or Ctrl, then click)

Watch a UML class diagram tutorial

Read about UML class diagrams

Watch Lucidchart basic tutorials