Célie Pierre
Mr. Knupp
Intro to Comp Prog II
May 6, 2019
Course Project Final Submission

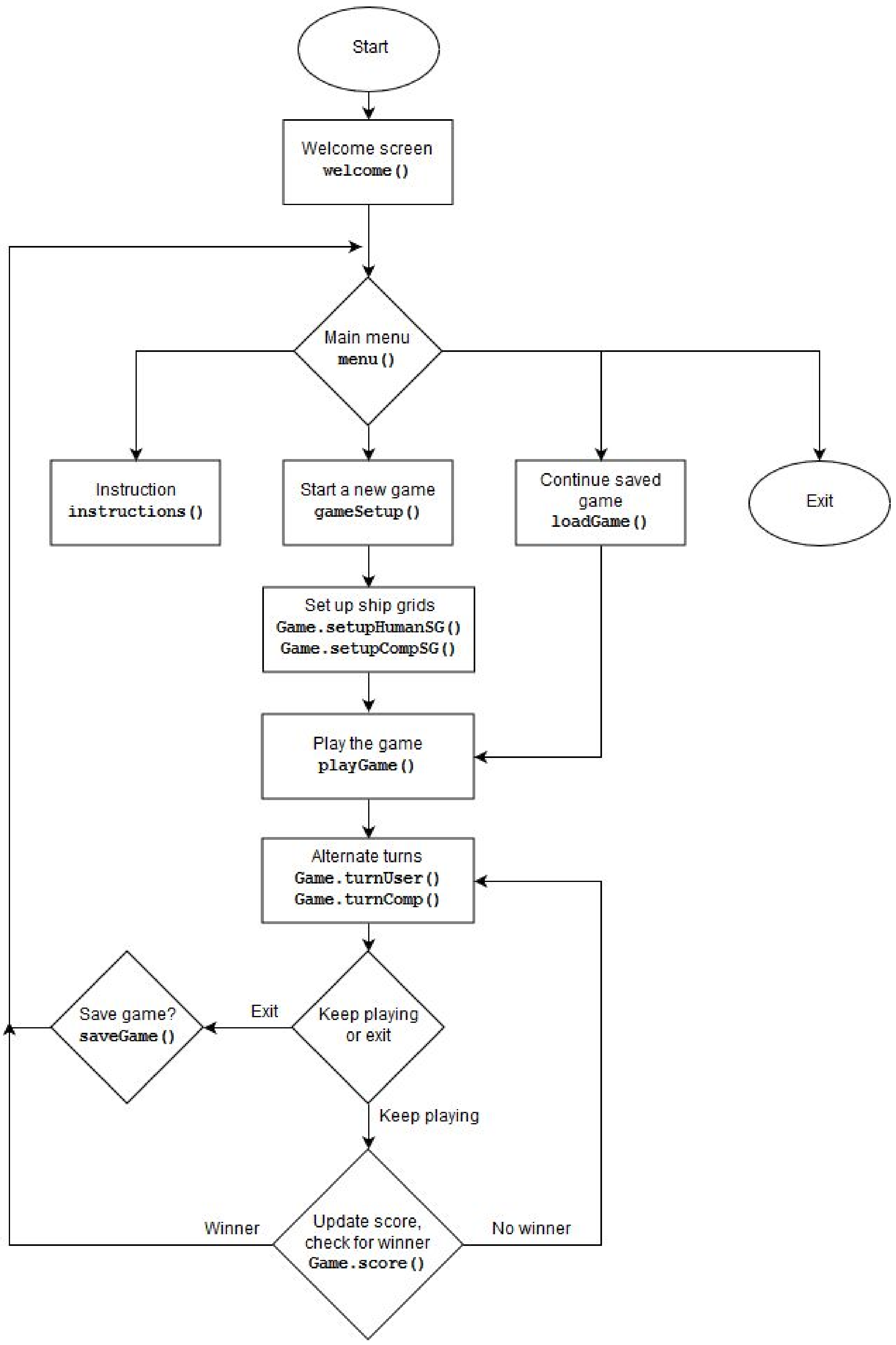
<u>Game Rules</u>

- Each player will set up their own ship grid
- Ships are arranged either horizontally or vertically, cannot overlap, must be completely on the board, and cannot be moved once the game has started
- Turns alternate between players
- Each player has 1 guess per turn
- Guesses consist of 1 letter (A-J) and 1 number (1-10) from the grid
- If you hit a ship, the hit ship's name is displayed and the player's guess grid is updated with an H
- If you miss, the player's guess grid is updated with an M
- A ship is sunk if all of their spots are hit
- Winner: The first person to sink their opponent's entire fleet of ships

<u>User Requirements</u>

- Players: 1 human player vs. 1 computer player
- Human will select from the main menu: review instructions, continue a previously saved game, start a new game, or exit
- Human will arrange each of their ships on their own 10x10 ship grid; grid will display each turn
- Computer will randomly arrange each of its ships on its own 10x10 ship grid; grid will be hidden
- Turns will alternate between human and computer
- Human will enter a guess each turn; their guess grid will update with either a hit or miss
- Computer will randomly select a guess each turn
- Human has the option to exit and/or save and exit each turn
- Game will continue until one player has sunk all of their opponents ships
- Ships: Carrier, length 5; Battleship, len. 4; Cruiser, len. 3; Submarine, len. 3; Destroyer, len. 2

Program Flow Chart



List of Functions

`main()`	Where the main portion of the program is held
`welcome()`	Displays the welcome screen
`menu()`	Displays the menu, allows user input for menu selection
`instructions()`	Displays instructions including ship setup, gameplay directions, formatting, and exiting the game
`loadGame()`	Loads a previously saved game from a data file
`saveGame()`	Provides the option to save the current game then writes that data to an output file
`gameSetup()`	Sets up a new game. Allows the users to arrange their ship grid, randomly sets up a computer ship grid.
`playGame()`	Where the game is played. Loops between human and computer player until a winner is declared.

Detailed Project Requirements

For this Battleship project, I tried to keep in mind coding best practices. I wanted to make sure that each variable, function, and object was self describing. There were a few places where I used very short names, such as “vh” for vertical or horizontal, but within the context it was used I would still consider it self describing. I say this because its first use was for user input where v and h were explicitly spelled out (`vh = input('Place vertically (v) or horizontally (h): ')`). Multiple variables included abbreviations such as SG and GG for ship grid and guess grid respectively, such as `humanSG`, `compGG`, and `setupHumanSG()`.

I also kept in mind my use of white space and indentation. I added an extra line between functions and objects, and sometimes within for organizational purposes. I made sure my indentation aligned properly, both for the benefit of a functional program but also so it was easier to read.

I chose to use Object Oriented Programming with this project. The BattleShip.py file is where the main program operates. The `main()` function starts the program, there’s a quick `welcome()` function, then the rest of the program takes place in a menu controlled while loop. The heavy lifting is done with the Ships.py file. That is where the Game class is located. It includes objects for both game setup and game play.

I used lists quite a bit throughout the program. All of the grids are held within lists. The ships and their data (their names, abbreviations, and lengths) were organized in a dictionary. I originally had the ship information in a list so it appears in both a list and dictionary data structure (more on this under 2.0 Release).

The user has the option to both save a current game and load a previous game. These options include the use of external files. To save a game, the current game data is written to a data file through the use of the pickle module. Loading a game also uses the pickle module and same data file.

I did quite a bit of testing and included appropriate error trapping. I added try/except blocks where any possible errors may occur. If an exception occurred within these blocks, I chose to include where the error occurred and to print the actual error message without stopping the program or causing it to crash. I also accounted for user input errors through the use of if/elif/else blocks, try/except blocks, validation objects, and the option to re-enter valid input through the use of while loops.

2.0 Release

There's so much that I'd like to see added to this program. First, I'd love to make the artificial intelligence smarter. Currently, the human has the upper hand because they can use their own logic. If you make a hit, it makes logical sense to choose an adjacent target for your next move until the ship has sunk. This seems easy enough to add a bit of logical decision making for the computer player but would take some trial and error.

Next, I would've liked to include a GUI but wasn't comfortable enough with the coding to implement that in this first version. It'd be a fun 2.0 add! I'd also like the ability to save more than 1 game and be able to choose from a list of saved games, probably just 3.

The other thing that you'll notice in this current version is that there is both a `shipsList` and a `shipsDict`. I started out with the list then decided on the dictionary. The game was pretty much done when I made that decision and worked better with the list in some places and the dictionary in others. I'd switch to just one or the other in 2.0, probably the dictionary.

I also had an issue with copying an empty grid which is why you'll see each grid listed in full. I'd work on that more until I had one empty grid that was able to copy properly. I tried a lot of different ways, I'm sure I'm just missing something simple. I listed some of my failed attempts in more detail in my Iteration 5 comments.

It'd be fun to include an option for human vs human - maybe even computer vs computer! I'd like to add a Salvo option for the game, too. In general, I'd like to simplify the code a bit more and improve the quality of the game itself. Overall, I believe version 1.0 works pretty well.