Pixel Adventure - Final Report-183773

Problem Statement

Pixel adventure is a 2D adventure game that takes the user through a story of saving the world, while battling many monsters. The game uses simple turn based battles to allow the user to defeat enemies. The user can do this to gain experience, level up, gain gold and complete missions. Enemies get stronger as the story progresses but the user also has the opportunity to get stronger via leveling up and purchasing items. Each level up comes with a boost in user AD (Attack Damage) and HP (Hit Points). Items serve many functions; some are necessary for entering particular areas, some give health boosts while in battle and others provide AD or HP boosts. Furthermore, Locations you can visit are locked based on story completion to encourage following the story. Since this game is unlikely to be finished in one sitting, the user has an option to save the game by visiting a bed. This save game can be restored from the start menu and can also be edited manually to change user profile by editing the users txt file.
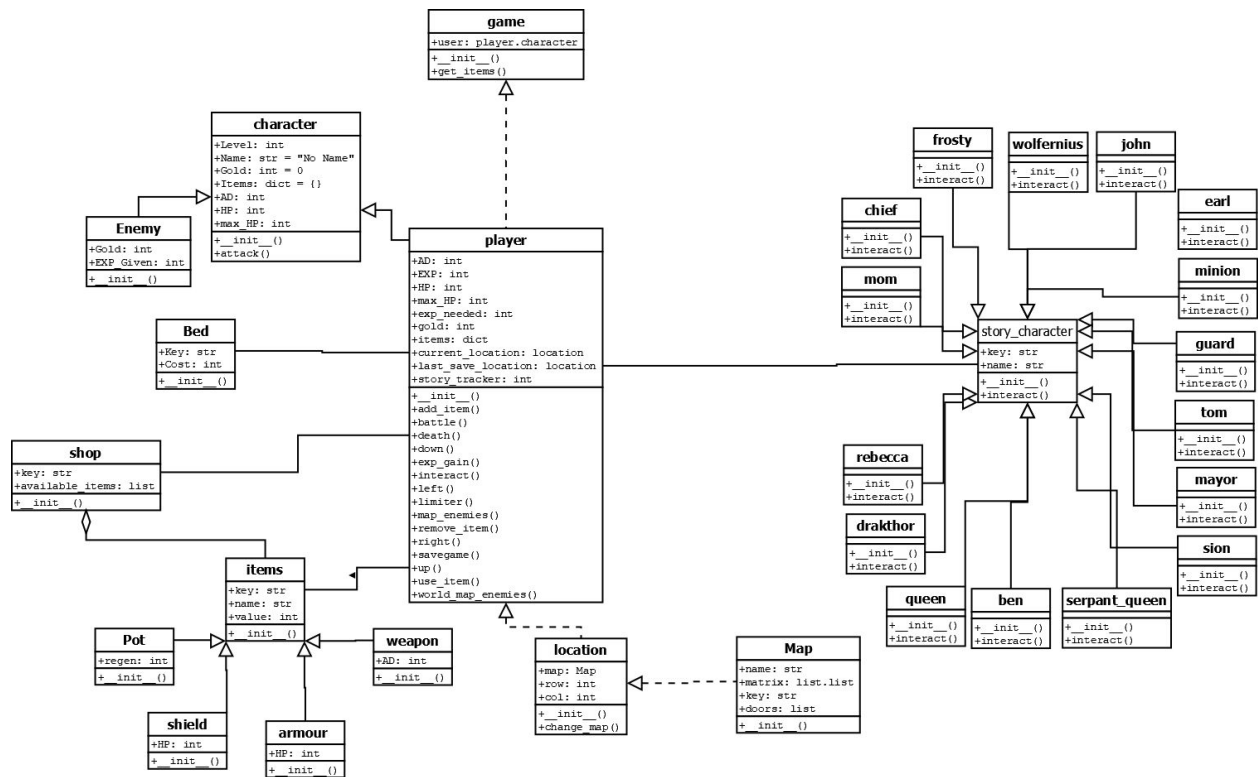
Running the Program

The program was developed in visual studio code using python 3.8.3 but should function just fine on any other IDE. I recommend running from the command line using "python3 main.py" or "python main.py". Also ensure that the tkinter and pillow libraries are installed. The window size is fixed and requires a minimum screen resolution of 720p. The program was developed in linux but has been tested and is working on windows.

Getting Started

To start the game choose whether to load a previous game from save game(txt) or start a new game. Once met with a blank screen press any of "wasd" to get the map to appear. If at any point the game has displayed a message and the user is unable to move, press return/enter to continue. This should allow the user to move again.

My peer review suggested I include a readme file to make it easier/clearer to startup the game. Thus, the above, along with some additional information is available in the included readme.md file.

Class Diagram

game
+user: player.character
+__init__()
+get_items()

character
+Level: int
+Name: str = "No Name"
+Gold: int = 0
+Items: dict = {}
+AD: int
+HP: int
+max_HP: int
+__init__()
+attack()

Enemy
+Gold: int
+EXP Given: int
+__init__()

Bed
+Key: str
+Cost: int
+__init__()

shop
+key: str
+available items: list
+__init__()

player
+AD: int
+EXP: int
+HP: int
+max_HP: int
+exp_needed: int
+gold: int
+items: dict
+current_location: location
+last_save_location: location
+story_tracker: int
+__init__()
+add_item()
+battle()
+death()
+down()
+exp_gain()
+interact()
+left()
+limiter()
+map_enemies()
+remove_item()
+right()
+savegame()
+up()
+use_item()
+world_map_enemies()

items
+key: str
+name: str
+value: int
+__init__()

Pot
+regen: int
+__init__()

weapon
+AD: int
+__init__()

shield
+HP: int
+__init__()

armour
+HP: int
+__init__()

location
+map: Map
+row: int
+col: int
+__init__()
+change_map()

Map
+name: str
+matrix: list.list
+key: str
+doors: list
+__init__()

frosty
+__init__()
+interact()

wolfernius
+__init__()
+interact()

john
+__init__()
+interact()

chief
+__init__()
+interact()

mom
+__init__()
+interact()

earl
+__init__()
+interact()

minion
+__init__()
+interact()

story_character
+key: str
+name: str
+__init__()
+interact()

guard
+__init__()
+interact()

tom
+__init__()
+interact()

rebecca
+__init__()
+interact()

mayor
+__init__()
+interact()

drakthor
+__init__()
+interact()

sion
+__init__()
+interact()

queen
+__init__()
+interact()

ben
+__init__()
+interact()

serpant_queen
+__init__()
+interact()

[Also available as a separate png file]

I made a few changes to the class setup from the part one submission.

- From my feedback from my tutor I removed all my enemy subclasses and created a dictionary with names for enemy class to access. This makes sense as the code was repeated in all of these classes, all but the name and level.

- I also decided to move some methods into the player class. The interaction methods for shop and bed inside location class seemed better suited to the player class as it is something the player does. This also helped make my code more OO pure as now shop and bed do only one thing but do it well.

- Furthermore, I removed the fight class as again this seemed better as a method for the player class as again it is something the player does.

- A small but meaningful change was the addition of a penalty for death. My peer reviewer suggested it should not be free for the player to die and as such I implemented a 10% gold cost to the player's death.

- The biggest addition was the story_characters class and all of its subclasses. To create a playable story I created this class to represent story objects that the player can interact with on the map. I made this class abstract, with the abstract method "interact" to ensure all subclasses can be interacted with. These classes use the player attribute "story_tracker" to determine what response the interaction will provide. "Story_tracker" increases as the player progresses through the story. Since many of these responses include logical statements, it would have been difficult to implement these from objects of one class. Thus, I found this approach to be most suitable.

- Finally, I also decided to merge the new_game and load_game classes into one game class. I noticed that they both do the same thing and the logic that determines which one to go with could be included in the constructor for the new class. I also moved some functions from the "main.py" to this class as they also include logic used for load_game.

GUI
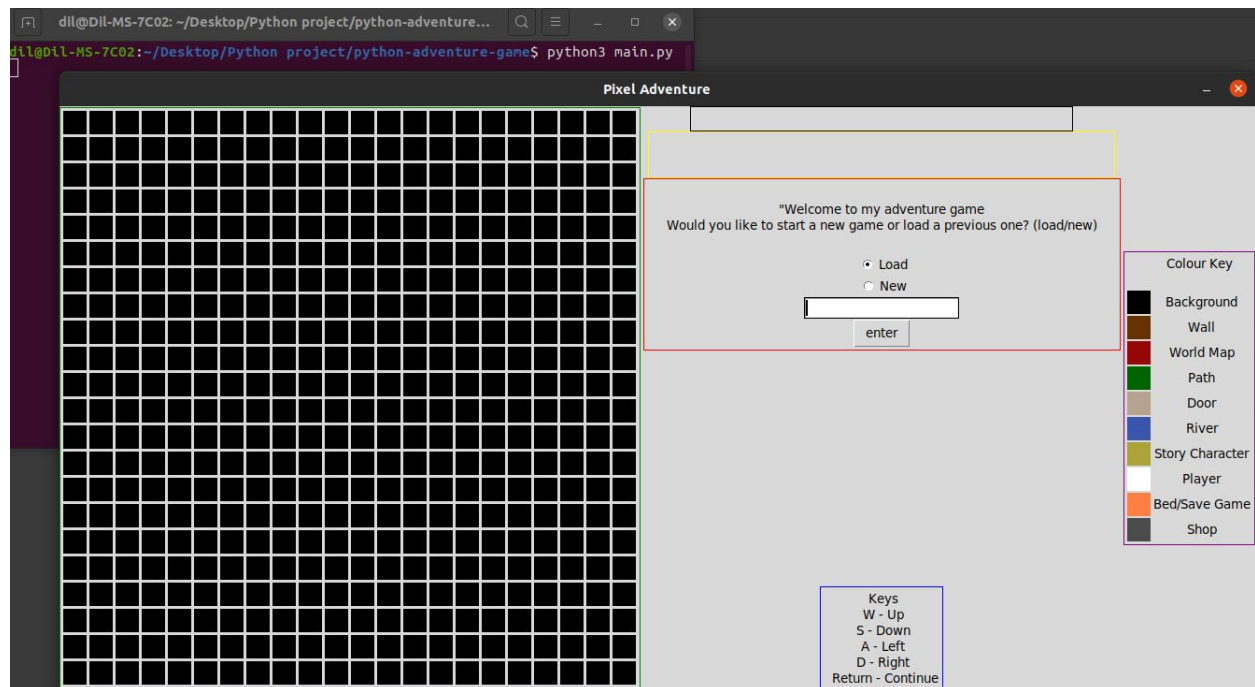
[Image of startup screen]
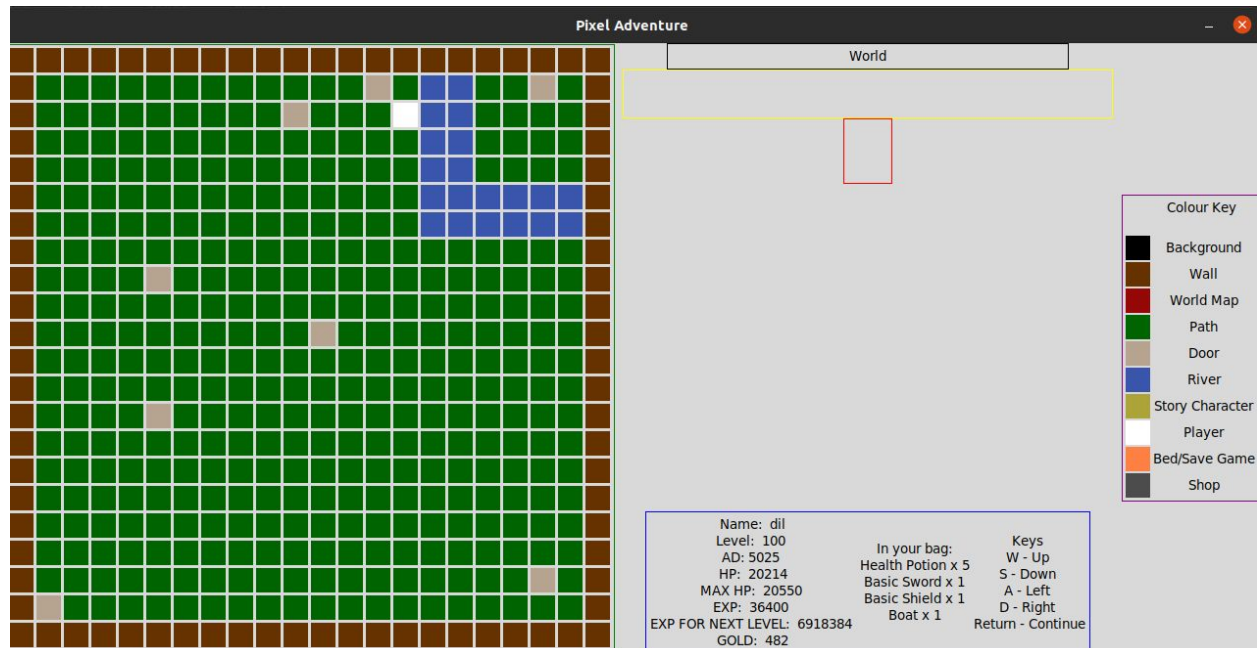
[Image of Gameplay]

I wanted to represent my map using squares for each element of my list of lists. I decided to use the grid structure of Tkinter to do this. I created labels of different colours and attached them to the grid but the size was an issue. As such I downloaded 25x25 pixel solid colour images and using tk.PhotoImage attached them to the grid. I started by attaching only the black squares as this is the standard background.I used these colour blocks to keep the program simple. I decided to use key-bindings on "wasd" as these felt most natural for gameplay.

I created a class called "grid_element". This was to efficiently store all the data for a Tkinter grid position. This includes the row and column on the full grid, the translated position for the 7x7 grid and the translated position for the 10x10 grid. This combined with my show_map2() method allows the gui to display the correct map in the correct position along with displaying the user on top.

I created the map_keys() method to change the image displayed for particular objects and terrain based on the key part of the tuple (inside the list of lists). This is run for each show_map2() and ensures each element is displayed correctly.

I also created the battle_mode_changes() method to set the gui to battle mode. This displays an image of crossed swords to inform the user that there is a battle.
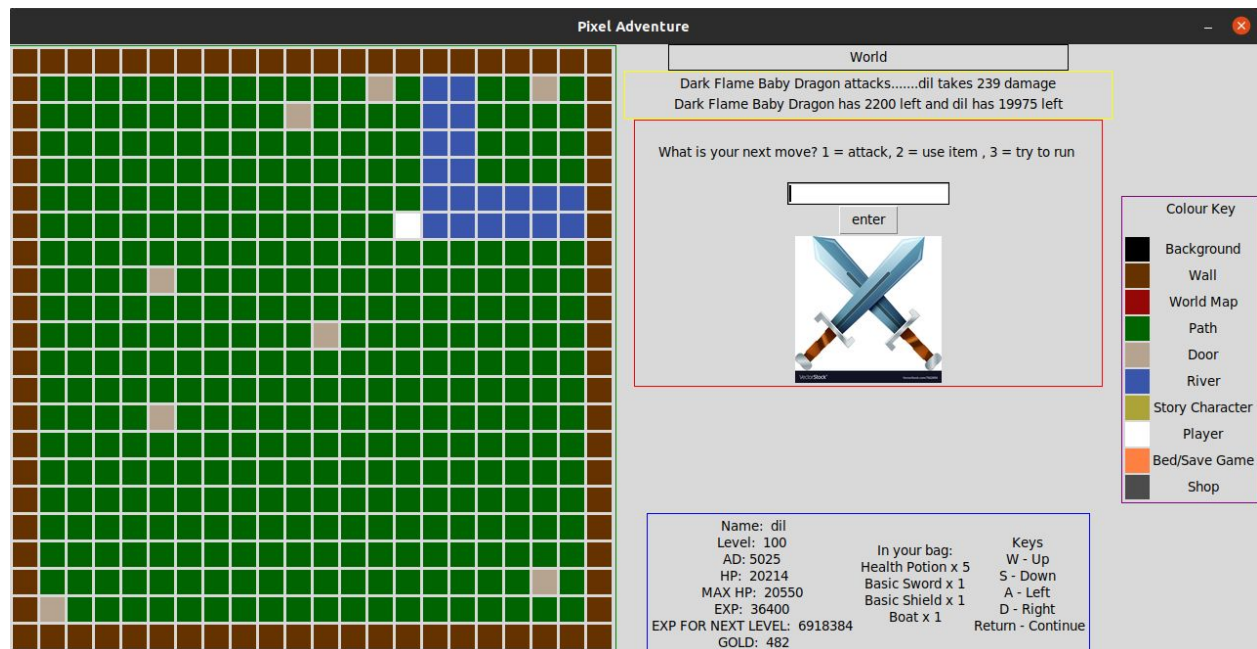
[Image of in battle gameplay]

To display game data I created a new print function "printer()". This prints the message to the gui and then waits for the user to press enter to continue. I had an issue with taking inputs but this was resolved by adding another parameter for printer "inp", which when True displays a input box but is set to False by default. I also have a "warning()" function that simply uses a tkinter message box rather than printing to the gui. I felt this would be more appropriate for certain situations, such as adding items to inventory or displaying an error message.
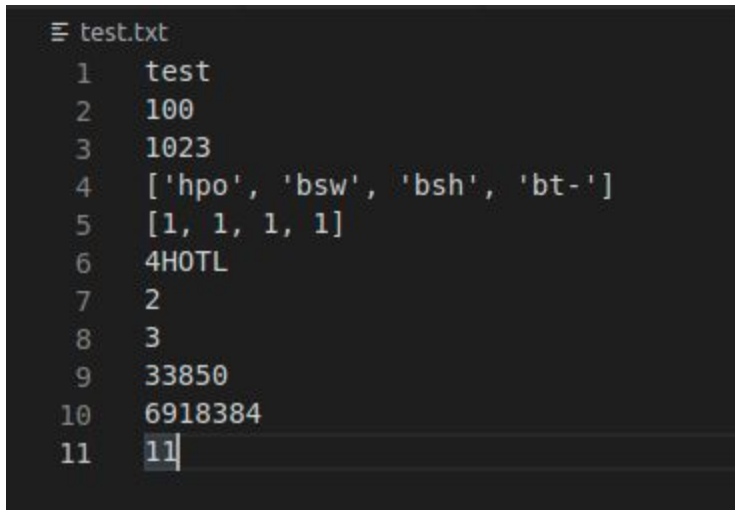
Finally, I organized my gui with multiple frames. The largest holding the map grid. The other frames hold useful information, including current map, game interactions, battle information, stats, instructions and a key for the different coloured squares.

Journey Log

My journey Log is not exactly the same as described in the specification but I feel it goes a step further. I have confirmed with my tutor that this is an acceptable substitute.

I have implemented a save game feature that records game progression and when used to restore, restores the player to the same point in story mode with all the same items and stats. This is done by recording all of the players vital attributes to a txt file given the players name. This is then parsed by the loading algorithm to load the game from the save point. This was complex due to needing to convert string data into objects. To solve this, I saved keys and looped through "all_maps" and "all_items" to find the corresponding and create a new game object with the same data as when saved. This also stores "story_tracker", as such if the player

decides s/he cannot finish the game in one sitting, s/he has the option to save and resume from the same point in the story.

```
≡ test.txt
  1   test
  2   100
  3   1023
  4   ['hpo', 'bsw', 'bsh', 'bt-']
  5   [1, 1, 1, 1]
  6   4HOTL
  7   2
  8   3
  9   33850
 10   6918384
 11   11
```

[Image of example save game data]

This was also extremely helpful in testing as I can easily adjust user profile, add and remove items, change location simply from the txt file.

Testing

A significant issue with my part one assignment was the lack of unit testing as mentioned by my tutor. This has now been rectified with thorough testing of all classes.

Unit Testing
I created unit test classes for all my backend classes to ensure correct functionality. A few of these open the gui, as it is integrated in these, and ask for user input. Most inputs can handle random inputs but the fight class will be most effectively handled with an input of 1, thus entering "1" for all inputs is the most effective way to complete these tests.

[Screenshot of tests completed with a PASS]

System Testing

I decided on the most important features for the user through playing myself and friends giving opinions. I have a pdf copy of the table containing the results from testing all of these. I made sure to include testing for user input errors as these can be critical if not handled correctly.

Task ref	Description	Inputs	Expected outputs	Actual Outputs	PASS/FAIL
1	Open new game	user inputs name	starts a new game with character with given name at level 1, in house, in home town	as expected	PASS
2	Savegame onto txt using bed	interact with bed and choose to continue	Game is saved and user is told to continue	as expected	PASS
3	Loadgame from savegame, 0 items	choose load at startup and type name associated with file	game resumes with user stats corresponding to those of previous save time and location	as expected	PASS
4	loadgame from savegame, with items (multiple pots)	same as above but with multiple items in particular multiple pots to see if the correct count is returned	same number of pots as before, other items in bag. Also check if weapon/shield/armour stats are accounted for in user stats	as expected	PASS
5	try load game with non existing savegame file	enter incorrect savegame name	prints error statement and starts new game with given name	as expected	PASS
6	check if stats increase on buying weapon/armour/shield	buy a stat boosting item	user stats should increase	as expected	PASS
7	purchase item of same type (weapon, shield, armour) and replace old one when told not possible	try to purchase item of same type (not pot)	given error message and asked if user prefers to replace	as expected	PASS
8	difference in hit points after a battle	check post battle Hp	value should be lower than before	as expected	PASS
9	win a battle	attack enemy to victory	enemy is defeated and user returned to map	as expected	PASS
10	level up	win battles to gain sufficient EXP	level up message should be printed and user stats updated	as expected	PASS
11	use health potion in battle	select option 2 in battle and use pot	hit points are increased by associated amount to type of pot	as expected	PASS
12	run away from a battle	select option 3 in battle and get away successfully	user returned to map	as expected	PASS
13	get blocked from running away from a battle	select option 3 in battle and fail to get away	user still in battle	as expected	PASS
14	get gold from killing an enemy	go into battle	user gold increased after battle	as expected	PASS
15	die in battle	run out of hp in battle	be taken back to last save location	as expected	PASS
16	incorrect input in shop	unplanned input in shop	warning message	as expected	PASS
17	incorrect input in bed	unplanned input in bed	warning message	as expected	PASS
18	incorrect input battle	unplanned input in battle	warning message + enemy attacks	as expected	PASS
19	move without enter after speaking with story character	try to move with pressing continue	unable to move	as expected	PASS
20	location indicator changes with location	change map	map name changes	as expected	PASS
21	cross river without boat	move to river without boat in items	unable to cross	as expected	PASS
22	cross river with boat	move to river with boat in items	able to cross	as expected	PASS

[Screenshot attached but pdf version also available in files]

Map Testing
The files contain a "Map Testing Version" of the game. This is to ensure all areas are accessible and not causing any issues/errors. This version has a few differences, notably:

Movement based on random generator
Removed waiting on print statements (removed myApp.contin= false)
User inputs are preset
Removed intro gui and set game = load_game(test) (fixed to game complete with boat to cross river)
Remove show_map via gui and use textui instead. - faster due to not needing to load images

There are 29 maps and as can be seen from the output file, all 29 were found through this test, with no errors.

[Image of completed Map testing - all maps reached in 49324617 moves]

Exception Handling, Class and method level Documentation

I have created an info message box for various situations and these include handling various user input errors as these are the most likely to cause system error. Through map testing I have tested for millions of movements (49million above) and seen no errors suggesting it is unlikely to have any critical errors.

I have substantial class and method level documentation that clearly describes classes, methods, parameters, and parameter types.