COMP9001 assignment2 report

Name: Yancheng Zhu
Unikey: yzhu7699
Question: talk about your development process
Answer to 1:
For game_engine.py, firstly, I follow the ED page's order to develop and consider my code structure. I just simply introduce my engine code.

1. for the __init__ function, besides some necessary attribute like self.score, self.fuel and so on, I also define some my own class attribute which are used in the other class function. For example , self.completeend is used to sign whether the game_state file is complete self.fuelstage is used in printing warning message. And so on.
2. For the import_state function: it's function is to import the data from the game_state file, and create corresponded objects which are used in run_game, also, if the data doesn't equal to the right format, program should raise some corresponding error.
   My idea or code structure of this function is firstly decide whether the file name inputted can be open. If it can be open, then begin check the data. I divide the check process to 6 stages.
   2.1 stage1: include line1~6.
   2.2 Stage2: check the asteroids lines
   2.3 Stage3: check the bullet count line
   2.4 Stage4: check the bullets initially exist lines
   2.5 Stage5: check upcoming asteroids count line
   2.6 Stage6: check upcoming asteroids lines
   At the end, check whether file is complete.
3. For the export_state function, it's purpose is write all the data to a game_state file.
   Just follow the format and use f.write(), be careful with the \n
4. For the run_game function, it's purpose is run game. I follow the ED order to structure my code.
   4.1 use self.player.action() to receive player input
   4.2 control the spaceship
   4.3 update the position of asteroids
   4.4 update positions of bullets
   4.5 deduct fuel and may print warning message
   4.6 detect collisions
   4.7 call self.GUI.update_frame()
   4.8 call self.GUI.finish()

about test and debug game_engine part:
firstly, I finish my space_object code and game_engine code, upload ED to test, only 9 passed, after I add wraparound code in collision and fixed some minor errors, I passed 27 I think. And in the test_game_engine_game1 testcase, seems very different output with the right output, finally, I find that I didn't notice the line provided self.player=player_class(). I just defined another line to get the player_class, which just instantiate it many times in game, after I fix this

problem, and some order of print messages, I just get the full passed.

Answer to 2:

In the wraparound for movement, it's easy to handle, by reassign the x and y coordinate of object, in 4 situation:

1. If(self.x<=0) it's the situation that object go over the left boundary
2. If(self.x>=self.width) it's the situation that object go over the right boundary
3. If(self.y<=0) it's the situation that object go over the up boundary
4. If(self.y>=self.height) it's the situation that object go over the low boundary

For example in situation 1: use code :self.x=self.width+self.x.

In the wraparound for collision:

Firstly, my idea is dividing 9 situations to discuss and deal with the x and y coordinate, but I think it's too difficult, it's not simple and efficient. Therefore, I analyze the key point of decide whether two objects collide with. The key point is the distance between these two objects.

Distance formula are constructed by the difference of x coordinate and difference of y coordinate. if two objects don't collide with, it means that the distance of two objects' center should more than the sum of the two objects' radius. So I just need to compare the least distance of two object's center with the sum of the two objects' radius. Then I can know whether they collide with.

Code as below:

```
minusx=min((min(self.width-self.x,self.x)+min(self.width-other.x,other.x)),abs((self.x-other.x)))
minusy=min((min(self.height-self.y,self.y)+min(self.height-other.y,other.y)),abs((self.y-other.y)))
distance=(minusx**2+minusy**2)**0.5
```

Minusx means the least difference of x coordinate with two objects

Minusy means the least difference of y coordinate with two objects

With these two least variables, I can get the least distance, compare it with the sum of the two objects' radius. If distance is larger, means not collide with, otherwise means collide with.

Answer to 3:

For my import_state method:

For the import_state function: it's function is to import the data from the game_state file, and create corresponded objects which are used in run_game, also, if the data doesn't equal to the right format, program should raise some corresponding error.

My idea or code structure of this function is firstly decide whether the file name inputted can be open. If it can be open, then begin check the data.

I use line=f.readline().strip("\n") to read data line by line with while(line) loop.

And I divide the check process to 6 stages.

stage1: include line1~6.

Stage2: check the asteroids lines

Stage3: check the bullet count line

Stage4: check the bullets initially exist lines

Stage5: check upcoming asteroids count line

Stage6: check upcoming asteroids lines

At the end, when line=None which means that I have read the last line of the game_state file, The loop will end, if program don't process the complete lines, which means that don't get enough data, the variable completeend would be False, so it will raise incomplete error.
When testing and debugging this part, firstly, I make some mistake in my code of change stage, because at right time to change stage is important for my code, so after I check my code again and again, I pass the testcase about import and export .

Question: talk about extending your game
Answer to 1:
If want to add one more spaceship. firstly, in the import_state function, I should check 2 spaceship lines rather than 1, and create spaceship2 object and append it to spaceshiplist. In the player file, should return a list include two list of Boolean variable, like [[True, False, True, False], [True, False, True, False]]. Of course in __init__part, I need to add another series of variable, for example, self.fuel2, self.score2, and so on. In export_state part, I need to write the data about second spaceship to the file. In run_game part, I need to control two spaceships to move or turn left or turn right or fire. About fire part, I need figure out which spaceship is fire and create corresponding bullets. I think about the detect collision and deduct fuel part can almost keep same as before. But when printing fuel warning message, I should point to which spaceship is warning. For player file in multiplayer mode, I think I need to write some code to avoid two spaceship collide with each other, for example, if the distance between two spaceships is close to the sum of two spaceships radius, I should write code to let one spaceship choose the action which keeps away from the other asteroid, and the other spaceship stay, until two spaceships are no longer close to each other.

Question: talk about your bot's strategy
Answer to 1.
Firstly, I analyze the rules, shoot one bullet cost two fuel, and every frame spaceship cost one fuel no matter how can I control the spaceship. Shooting one small asteroid we can get 150 score, shooting one large asteroid we can get 100 score. If spaceship collide with asteroids, minus 100 score. So if player want to get more score with the same beginning fuel. Spaceship should focus on saving time and shooting effective bullets. And avoid spaceship collide with asteroids.
I divide action method in player 4 parts.
Part1: choose the asteroid spaceship should move to.
I define a function named choose_asteroid(). In this part, I use the idea from greedy algorithm. Every frame I just choose to go for the asteroid which can give player most profit. I define profit as the score of the asteroid divide the distance between spaceship with the objects. By the way, I define distance_objects() function to return the least distance between two objects, because I need the distance between two objects in many part of my code. In the choose_asteroid function, I just traverse all the asteroids and return the target asteroid.

Part2: when the distance between spaceship and the target is more than shoot range, choose action that the action method should return.

I define a function named choose_action(). in the function, when choosing action, because no matter spaceship move or not move, spaceship will deduct one fuel every frame, so I only consider three possibilities of action in this part.

Action1: move forward

Action2: turn left and move forward

Action3:turn right and move forward

I use simulate these three action to choose best one and return.

Part3:when the spaceship enter the efficient shoot range, define a function named choose_fire_direction() to get the right fire direction, if spaceship's angle is very close to the right fire direction. Then fire.

Part4: when the spaceship enter the efficient shoot range, but spaceship's angle is not very close to right fire direction, just make action function to return right list make spaceship turn to the right angle..

Answer to 2.

As I answer in above question,

firstly, I traverse the incoming asteroids list in choose_asteroid function to get the target asteroid, depending on the distance between asteroids with spaceship and the obj_type of asteroid.

Secondly, according to the distance between target asteroid, decide to use choose_action function to return action or compare the right angle with the spaceship's angle, to choose turn or not, and fire or not.

Answer to 3:

If I know the upcoming asteroids, I think in my strategy, most parts would not change, but I can add one more special situation, I will traverse the upcoming asteroids list, and if find there are many consecutive asteroids in the upcoming list are very close, then sign the first of these asteroids named bonus asteroid. If the bonus asteroid shows in the asteroids list, the spaceship will go for the bonus asteroid to shoot. Because in this situation, player can save lots of fuel but shoot many asteroids.