Name: ______________________________

The midterm exam must be completed in class and within the allotted class time. All answers are to be submitted electronically. You will submit your answers on Moodle within a single `zip` archive, named `lastname.zip`. **You may not communicate with other students during the exam.**

Introduction to Python

1. For this question, submit your answers to `intro.py`.

   $\boxed{5}$ (a) Write a Python `for` loop to print all of the odd integers between 1 and 100, inclusive.

   $\boxed{5}$ (b) Write a function named `fizzbuzz(low, high)`. This function will iterate over all numbers between `low` and `high` (as a half-open interval including `low` but not including `high`). If the number is divisible by 3, print `Fizz`, if the number is divisible by 5, print `Buzz`, and if the number is divisible by both 3 and 5, print `FizzBuzz`.

Exploring Python

2. For this question, submit your answers to `explore.py`.

   $\boxed{5}$ (a) Write a function called `acceleration(n, t)`, which returns the acceleration for the following function:

$$a(n,t) = \begin{cases} t^3 + 2t + n, & \text{if } n < 0 \\ nt^2 - t, & \text{if } n \geq 0 \end{cases}$$

   $\boxed{5}$ (b) Write a function called `clip_health(health)`. This function should clip the health value such that it is always between 0 and 100, inclusive. Thus, if the health is greater than 100, the function will return 100. If the health is less than 0, the function will return 0.

3. For this question, submit your answers to `tank.py`.

   $\boxed{3}$ (a) Create a class called `PredatorTank`. It should have an initialization method (`__init__`) that takes in a `name`, `health`, `attack`, `position`, and `range`.

   $\boxed{2}$ (b) Create three tanks: `t1`, `t2`, and `t3`. You can name these tanks "T1", "T2", and "T3". Each tank will have a health of 100, an attack of 5, and a range of 7. The position of T1 is (0, 0). The position of T2 is (1, 2). The position of T3 is (11, 16).

   $\boxed{5}$ (c) Within this class, write a method called `fire(self, target)`, that fires upon another `PredatorTank`. It should subtract `attack` units from the target, but only if it's own health is greater than 20 (otherwise, the unit is too damaged to fire). If the unit is out of range, then the attack will not be successful (i.e, do nothing). The health of a unit can never fall below zero.

Pygame

4. For this question, you will add your changes to `simplegame.py`.

$\boxed{2}$ (a) Add logic to display `boy.png` at location $(100, 100)$ on the screen.

$\boxed{2}$ (b) Add logic to display `girl.png` at location $(300, 300)$ on the screen.

$\boxed{6}$ (c) When the user presses the `t` key, the `girl.png` image should toggle to `horn.png`. Pressing the key again will toggle the image back to `girl.png`.

Movement

5. For this question, you will add your changes to `rook.py`.

$\boxed{5}$ (a) Create two classes called `Pawn` and `Rook`. Both classes take in two arguments: a `name`, and a `position`. For example, a Pawn object and Rook object might be instantiated with code such as:

```
p = Pawn("P1", (5, 5))
r = Rook("R1", (3, 2))
```

$\boxed{5}$ (b) Add a `find(self, targets)` method to the `Rook` class, where `targets` is a list of Pawns or Rooks. This method will return an (x, y) tuple that indicates *how* the Rook should get to the closest (using Manhattan distance) Pawn or Rook. For instance, if the closest Pawn is at $(2, 5)$ and a Rook at $(1, 1)$, the function should return $(1, 4)$.

6. For this question, you will add your changes to `flee.py` in the `flee` directory.

$\boxed{10}$ (a) The current implementation of the `flee` function is `pass`, which means that it does not do anything. Modify the `flee` function so that the AI flees the player until it is 200 pixels away. Once it is 200 pixels away, the AI should stop moving.

World Representation

7. This question requires that you modify the `game.py` file in the `world` directory. This version of the game generates a random level of different size each time the game is run (often, these levels will be quite poor, and the AI can get trapped due to poor obstacle placement).

$\boxed{10}$ (a) Add logic to the game loop so that the user can press the "s" key (`K_s`) to toggle that tile on which they are standing. For example, if the tile that they are standing on is `'#'`, then the `level` dictionary at that location will need to be changed to `'.'`. This is a handy feature to see whether or not the AI can dynamically adjust their route.

$\boxed{10}$ (b) Add logic to the game loop so that the user can press the "c" key (`K_c`) to clear the entire board of all obstacles. The `K_s` key can then be used to customize the obstacles.

Decision Making

8. For this question, you will modify `game.py` in the `decision` directory.

   $\boxed{10}$ (a) The current decision tree prioritizes `health` (defense) over `fire` (offensive) when the AI is low on both. Modify the decision tree so that the AI is more aggressive. That is, the AI should continue `firing` at the opponent if it is in the same tile, even when it is low on health. Only when it runs out of ammo should it seek additional health.

   $\boxed{10}$ (b) Add a `berzerk` mode to the AI. The AI goes into `berzerk` mode when it has a health of less than 10. When this condition occurs, the AI is now able to successfully fire at the player when the player is a) in the current tile as the AI, as well as b) adjacent (up, down, left, right) to the current tile as the AI. When the AI is in `berzerk` mode, it performs double damage on the player. (An easy way to test this is by attacking the AI and not allowing it to obtain any health items).

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	20	
8	20	
Total:	100	