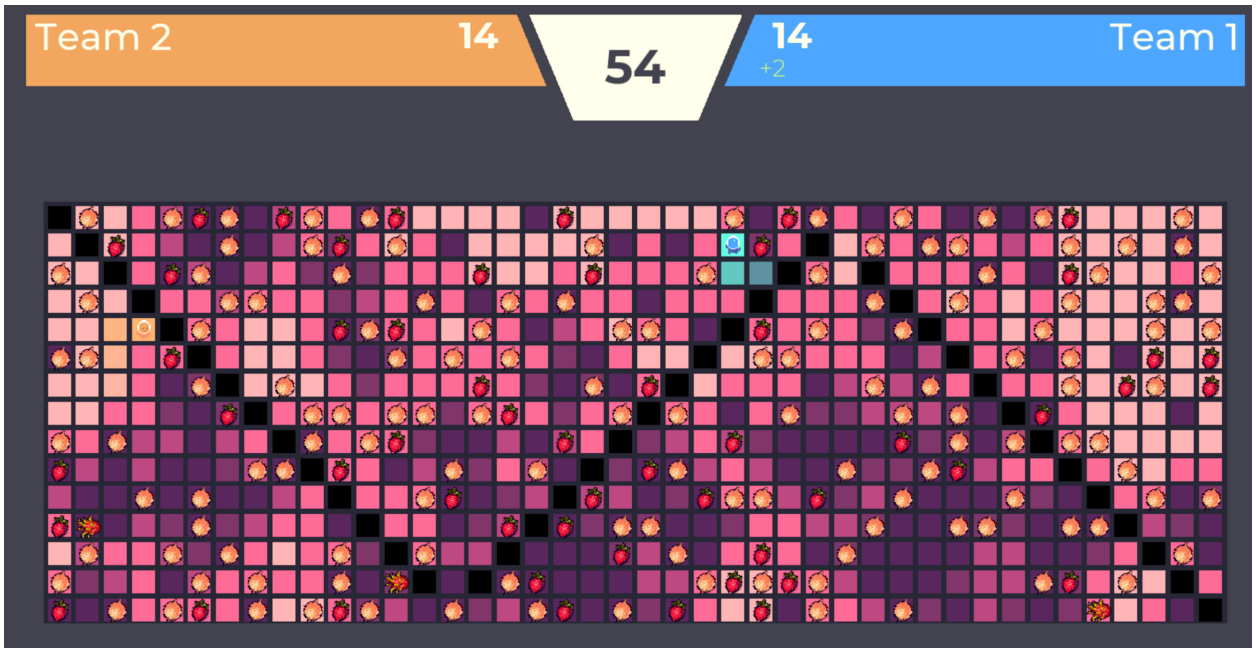
AI Arena 21

Game and Technical Documents

The Game

Outline

The game is played on a 2D rectangular map. Each cell is either blocked or free. Each map will have 2 spawn locations. Each of the players will be spawned by random at one of these locations at the beginning of the game. Different items are deployed onto the map and players should go around the map to collect them. Different items have different scores and the winner is the player who gets highest score by the end of the game.



Players

Each player has an ongoing score counter. In each turn, players can move to an adjacent cell (edge adjacent, not corner) that is not blocked. Players additionally have the option to rent two types of power-ups using their accumulated score: a **bike** or a **portal gun**.

Renting a bike will deduct 30 from your score and will give you the ability to move to a cell with a distance **less than or equal to 3** for the **next 3 turns**. Renting a portal gun will deduct 100 from your score and lets you teleport to any cell you wish on the map for **only that one turn**. You can't keep your items for future rounds! As soon as you purchase them the timer starts! 3 rounds for the bike and 1 for the portal gun.

Once a player enters a location on the map, they will collect all of the items located there and their score will be updated, unless the other player goes to that location at the same time too, in which case, an Auction will happen.

Items

A new batch of items is deployed on the map every 20 turns. There are 3 different types of items which will award different scores:

- Onion: has only 2 points because let's be honest they are onions
- Strawberry: has 10 points
- Dragon Fruit: has 30 points

Items' Heatmap

In the 20-turn leadup to the deployment of a new batch of items, more and more information is given to the player of where these items will spawn in the form of a heatmap. Every second turn, this heatmap becomes more and more accurate. In particular, on the (20 - 2x)th turn, the value at a particular square is calculated as follows:

- Consider all other squares which are d Hamming distance away (Ignoring collision).
- For each square which is $g \leq d$ distance away, and contains p points worth of items, we increase the heatmap value by $\frac{p}{(g+1)^3}$.

  So further away squares contribute less to the heatmap.

So as the items are closer and closer to spawning, the heatmap considers a smaller area when calculating its value.

Auction

When both players enter the same location on the map at the same time, or if 30 turns have passed since the last auction, they enter an auction for their location(s). Each player offers an integer bid and the player with the higher bid wins the auction. The winner gets to stay in their current location and teleports the other player to a location of the winner's choosing on the map. The winner then pays the floor divided average of the two bids.

So for example, if player A offers 10 points and player B offers 20 points, player B will remain in that location, pay 15 points, and teleport player A to some other location.

Players cannot bid higher than their available points. If both players bid the same amount, they will both be teleported to random locations on the map and the items will remain in the cell. If an invalid transport location is sent, the target player will be transported to a random location on the map.

Technical Implementation

*You can also find **in addition** to the explanations below, a quick demonstration of these explanations in this video.*

Getting Started

You will need Python (3.6+) and pip installed to run the game. Both of these are most likely already installed on your computer but if for some reason you don't have them installed, you can download Python from https://www.python.org/downloads/ (While running the installer, ensure you tick "Add Python 3.9 to PATH"). You can confirm you have them installed by running this command (any of the 4 variations that show **python3** at the end of the version output is fine):

```
pip --version
pip3 --version
python -m pip --version
python3 -m pip --version
```

Once you have it installed, run this command to install the AI Arena game (with the correct variation of running pip from above):

```
pip install ai-arena-21
```

Running the Game

To make sure your installation (and later on, your code) is working, you can run the game using this command:

```
aiarena21 random_source random_source
```

This should start the game with two random bots going around the map. You can change the map by setting `-m` parameter. You can either use one of the maps made for you (1.txt, 2.txt, … to 7.txt) or may choose to test your bot on your own map. To create a map you need to create a file and put the map in there. It should be a rectangle and each cell should be either . (for free cell), # (for block cell) or S (for spawn point - spawn points are optional). This is an example map:

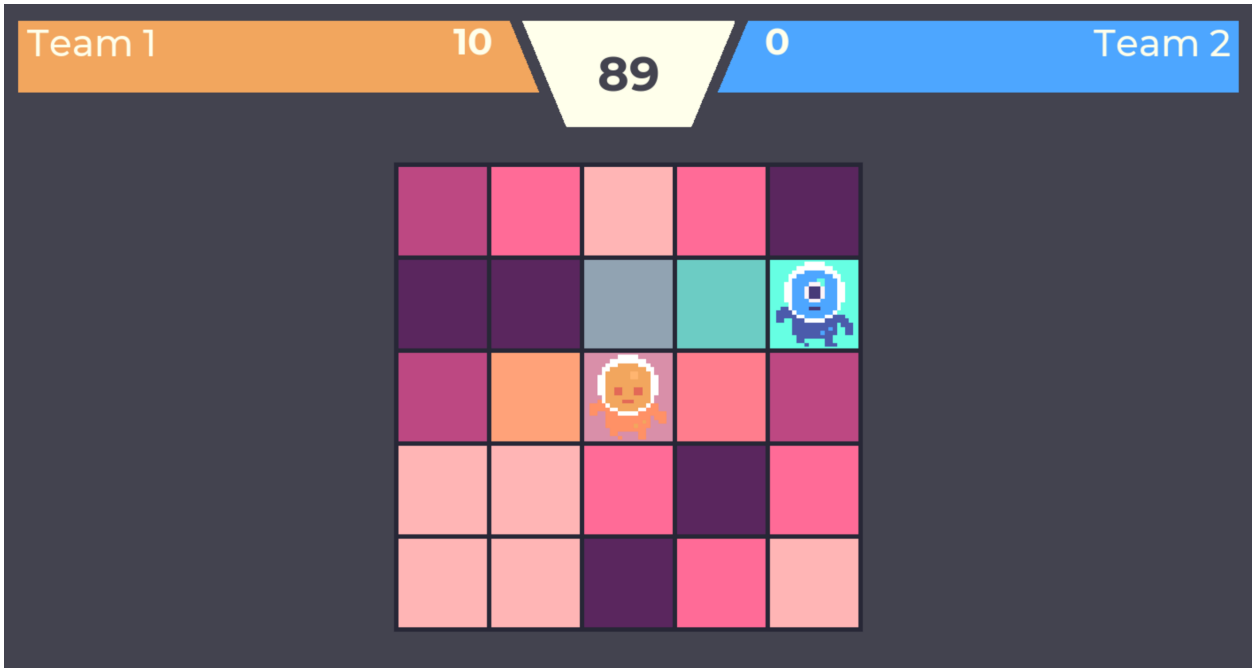
```
.......
..###..
.......
S.S....
```

Save the file with some name. For this example we assume you have named your file `my_map.txt` which then you can pass to the game like this:

```
aiarena21 random_source random_source -m my_map.txt
```

Note that the map file should be in the current directory from which you are running the game.

```
(venv) ➜  ~/Desktop/AIArena21 ls
my_map.txt
(venv) ➜  ~/Desktop/AIArena21 cat my_map.txt
.....
...S.
.....
.S...
.....
(venv) ➜  ~/Desktop/AIArena21 aiarena21 random_source random_source -m my_map.txt
pygame 2.0.1 (SDL 2.0.14, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
Server Started
Team Team 2 connected...
Team Team 1 connected...
```



Implementing Your Bot

All of the source code for your bot should be inside a single python file. This file needs to have at least these 3 functions defined or the game won't be able to use otherwise.

- **play_powerup**
  This function will be called at the beginning of your turn. This is how your program will be asked to say which power up it wants to purchase. You can purchase at most 1 power up per turn.
  It should return exactly one of these strings: *'bike'* - *'portal gun'* - '' (empty string for no power up purchase).

- **play_turn**
  This function will be called after you have chosen your power up. Here you should say where you want to go. This function should return a string in this format: **'*6,11*'** where 6 in this example is the row you want to go and 11 is the column you want to go. Note that there is no space in the string anywhere.

- **play_auction**
  This function will be called whenever your bot is drawn into an auction whether because both players have entered the same location or because there has not been an auction for 30 rounds. In this function you should say how much you would like to bid in order to keep your location and not be transported somewhere else. Refer to the *Auction* section from above for more info. This function should return an **integer** indicating your wager.

- **play_transport**
  This function will be called whenever you have won an auction and now get to transport your opponent. You should return a **string** here with the exact format of the *play_turn* function, indicating where you would like to transport your opponent.

All of these functions use the same set of arguments:

- **`game_map`**: this will be a `Map` object and it will contain the information of the map. You can use these attributes on this object:
  - `game_map.rows ->` the number of rows in the map
  - `game_map.cols ->` the number of columns in the map
  - `game_map.get(A, B) ->` returns the cell at Ath row and Bth column of the map. Which can be either a free cell, a blocked cell or a spawn location.
  - `game_map.is_free(A, B) ->` returns either true (if the passed location is free i.e. is not blocked) or false.
  - `game_map.BLOCK_CHAR ->` this is equal to '#' and can be used to check whether a cell is blocked or not.
  - `game_map.FREE_CHAR ->` this is equal to '.' and can be used to check whether a cell is free or not (only in this context free and spawn are different. In all other places a spawn location is considered as a free cell).

    - `game_map.SPAWN_CHAR ->` this is equal to ‘S’ and can be used to check whether a cell is free or not.

  Note that not all maps have spawn locations.
- **`me`**: this will be a Player object and it will contain the information about your player in the map. You can use these attributes on this object:
    - `me.name ->` to get your team name.
    - `me.score ->` to get your score.
    - `me.location ->` returns a tuple with 2 elements, first of which is the row your player is at and the second one is the column.
    - `me.bike ->` a bool which will be true if you currently possess a bike and will be false otherwise.
    - `me.portal_gun ->` a bool which will be true of your currently possess a portal gun and will be false otherwise.
- **`opponent`**: this is also an object of type Player and has all of the attributes explained above. So you can see your opponent’s locations, score and power-ups during the game.
- **`items`**: this will be a 2-d list with the exact shape and size of the game map. Each cell of this 2-d list will be an integer which is the sum of the items (fruits) in that location. If no items, the sum will be 0.
- **`new_items`**: this will be a 2-d list of the same shape and size of the game map, containing only the new items deployed on the map just before that turn. Each cell of this map will be a list of strings. Each string in each cell is the name of the item deployed to that location in that turn (either **Onion, Strawberry, DragonFruit** - case sensitive).
- **`heatmap`**: a 2-d map with the same shape and size of the game map where each cell is a real number as described above in the *Items’ Heatmap* section.
- **`remaining_turns`**: an integer letting your program know how many more turns of the game are left.

This is the source code of a bot that does everything randomly (you will also receive the .py file of this source).

```
from aiarena21.client.classes import Player, Map

import random


def play_powerup(game_map, me, opponent, items,
new_items, heatmap, remaining_turns):
    return random.choice(['bike', 'portal gun', ''])


def play_turn(game_map, me, opponent, items,
new_items, heatmap, remaining_turns):
    dx, dy = 1, 1
    while dx*dy != 0 or dx+dy == 0:
        dx = random.randint(-1, 1)
        dy = random.randint(-1, 1)
    new_row = me.location[0] + dx
    new_col = me.location[1] + dy
    return f'{new_row},{new_col}'


def play_auction(game_map, me, opponent, items,
new_items, heatmap, remaining_turns):
    return random.randint(0, min(opponent.score,
me.score))


def play_transport(game_map, me, opponent, items,
new_items, heatmap, remaining_turns):
    return f'{random.randint(0,
game_map.rows-1)},{random.randint(0,
game_map.cols-1)}'
```

When inside the directory containing your source code, you can run the game playing with that source code by passing its file name **with or**

without the .py extension to the command explained above. So if you have implemented your bot in a file called **my_source.py** the command to run the game against random_source would be:

```
aiarena21 random_source my_source
```

Recording and Analysing games

Not opening a window

If you want to run games quickly, without the visual window being open, you can!
Simply add the **--no-visual** or **-nv** tag to your call to aiarena21:

```
aiarena21 random_source random_source -nv
```

Replay files

But what can we do with this? You might've noticed that a file called **replay.txt** is created whenever we run aiarena21. You can use this file to analyse every action and event in the game.

You can specify the replay file location with the **--replay** or **-r** flag to replay and rewatch one of your games and extract insights:

```
aiarena21 random_source random_source -r
my_replay.txt
```

But what's in this replay file anyways?
Each line of replay.txt is the string [Replay] followed by a JSON object.

Line 1

On the first line we have a JSON object describing the initialisation process of the game:

- **`grid`:** The map that is being used for this game
- **`scores`:** The initial scores of each player
- **`teams`**: The team names
- **`spawn`**: The spawn location of each player
- **`total_rounds`**: The number of steps the game will run for

Other lines

Every other line is then a JSON object describing the next tick of the game

- **`heatmap`**: Heatmap data for the map
- **`positions`**: new_pos is the updated position of the player, delta is the distance they've moved this tick (different for wagers)
- **`items`**: List of items in each grid square
- **`bike`**: Whether each player has the bike activated (Stays true while the bike is active)
- **`teleport`**: Whether each player has the teleporter activated
- **`scores`**: Current scores
- **`remaining_rounds`**: How many rounds are left to simulate

Final line

This will continue until **`remaining_rounds`** is 1, and then **`[Replay]{"type": "finish"}`** is placed at the end of the replay file.

Use this information to improve and analyse your bot however you please!