CPSC 4160/6160 Fall 2022

PROJECT 3

Due: In-class demonstration on November 22, 2022; code delivered to handin by end of day, November 22, 2022

DESCRIPTION:

This project is to build more entities and actions in your game engine, and implement a game called "Cannon Fodder". You will be using engine content that you have already built, and also build new ones, particularly in the physics engine. The purpose of the game is to verify and demonstrate the performance and execution of the game engine, with the new elements added in this project. This project is worth 15 points, divided up broadly into three categories:

1. Does it work and are the specific requirements of the assignment implemented? (5 points)
2. Was the demonstration informative about the strengths and weaknesses of your project? (5 points)
3. Is the code clean, organized, and demonstrative of pride of craftsmanship, and easy to execute? (5 points)

On November 22, you will submit this project for grading in two ways:

1. In-class demonstration: You will have 1-2 minutes with either the instructor or TA to demonstrate, live, your game playing. You should show all of the features of gameplay asked for in the description below. This is your chance to present your project in the best light possible.
2. By the end of the day, November 22, you must submit the code for your project to the SoC handin system. The specifics for doing this are posted on the website for the class. Pay careful attention to this task: many students who were very competent have messed up this task and suffered point loss. We will expect to be able to run the game on our own machines, as well as inspect the code.

GAME ENGINE

As we have discussed in class, our python-based game engine must have the following organization of directories (`gamey` is the name of my game engine code - feel free to give yours any name that you like, while retaining this directory organization inside):

```
gamey
├── assets
│   ├── sounds
│   └── textures
├── engine
│   ├── actor
│   │   ├── action
│   │   └── entity
│   ├── asset
│   │   ├── action
│   │   └── entity
│   ├── crowd
│   │   ├── action
│   │   └── entity
│   ├── enviro
│   │   ├── action
│   │   └── entity
│   ├── fx
│   │   ├── action
│   │   └── entity
│   ├── physics
│   │   ├── action
│   │   └── entity
│   ├── play
│   │   ├── action
│   │   └── entity
│   ├── render
│   │   ├── action
│   │   └── entity
│   ├── sound
│   │   ├── action
│   │   └── entity
│   ├── story
│   │   ├── action
│   │   └── entity
│   ├── ui
│   │   ├── action
│   │   └── entity
│   └── utility
│       ├── action
│       └── entity
├── python
└── templates
```

Do not forget that inside each directory, there needs to be a valid python file called “__init__.py”. An empty file is valid, but you can also use it to create utility or factor functions, for example.

If you find that you want to implement more actions and/or entities than described here, feel free to do so. However, you should be careful whether or not more are actually needed for the game that must be implemented. More can certainly be useful in the long run for the game engine, but pace yourself and make sure you get the assigned project done.

Final word of advice, paraphrasing Gandalf: “Keep it simple, keep it safe”.

CANNON FODDER GAME

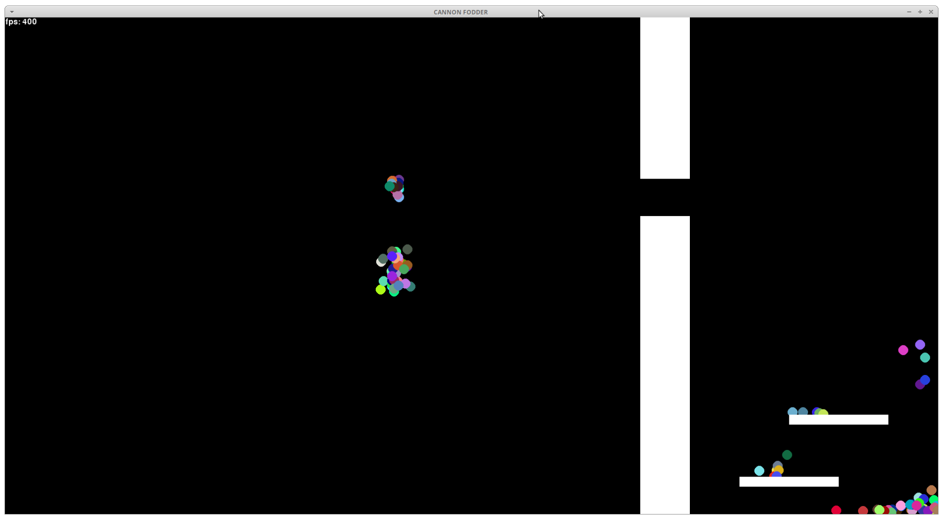
The Cannon Fodder game provides several different opportunities and challenges. The entirety of the physics engine, as shown in lectures, must be implemented. The physics entities interact with other entities via appropriate actions. The other entities and actions should already exist in your game engine, based on lectures and previous projects. If not, you will need to implement them.

The nature of this game is as follows: A collection of 100 particles exist at the start of the game. The particles experience gravity, spring, and drag forces. They also collide with the boundaries of the window frame and with two rectangles positioned toward the right hand side of the window, to form a narrow passage connecting the left and the right. The particles start on the left side, with net forces of gravity and spring, so that they oscillate around their collective center of mass. Initially the particles are randomly distributed in the left side of the window, with initial velocities making them move toward the right. When a particle passes through the narrow passage and enters the right side, it no longer feels the spring force and no longer contributes to the center of mass calculation in the spring force. It also feels a drag force as well as the gravity force. As a result, the particle drops downward and slows, and eventually stops moving. Sometimes a particle may be so energetic that, before it can fall very far, it happens to pass back out of the passage and enters the left. When that happens, the drag on that particle is lifted and the spring force resumes on it.

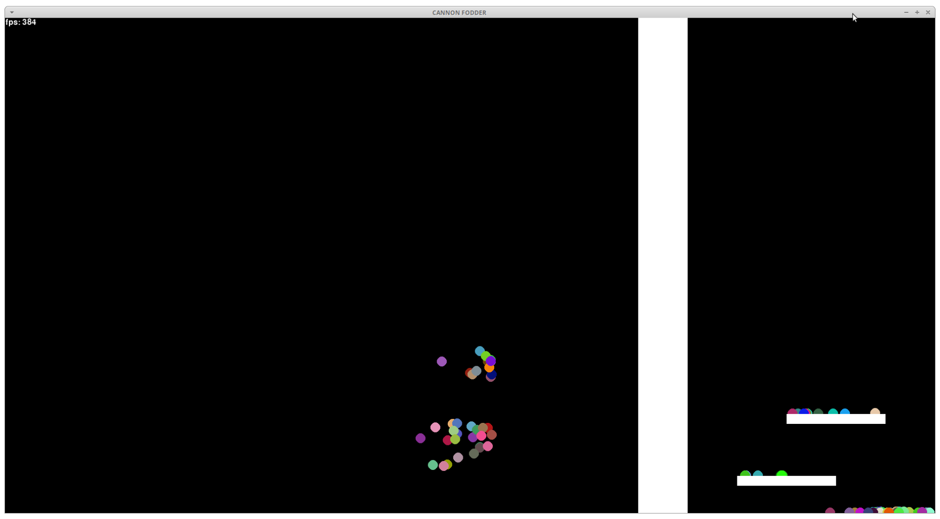
Each particle is associated with a circle actor. So there are 100 circle actors. The position of each circle actor is tied to the position of each particle. The circle actors are drawn to the screen, so that we can see the motion of the particles.

There are two more rectangular colliders on the right side below the level of the passage, for the descending particles to collide with. Some particles will bounce off of these colliders, and some will not have enough energy to move and will instead stay in place.

A screenshot of the game in play is here:



A timer starts at the beginning of the game, and after 9 seconds (9000 milliseconds) the passage gap disappears and the two sides are forever separated, like this:

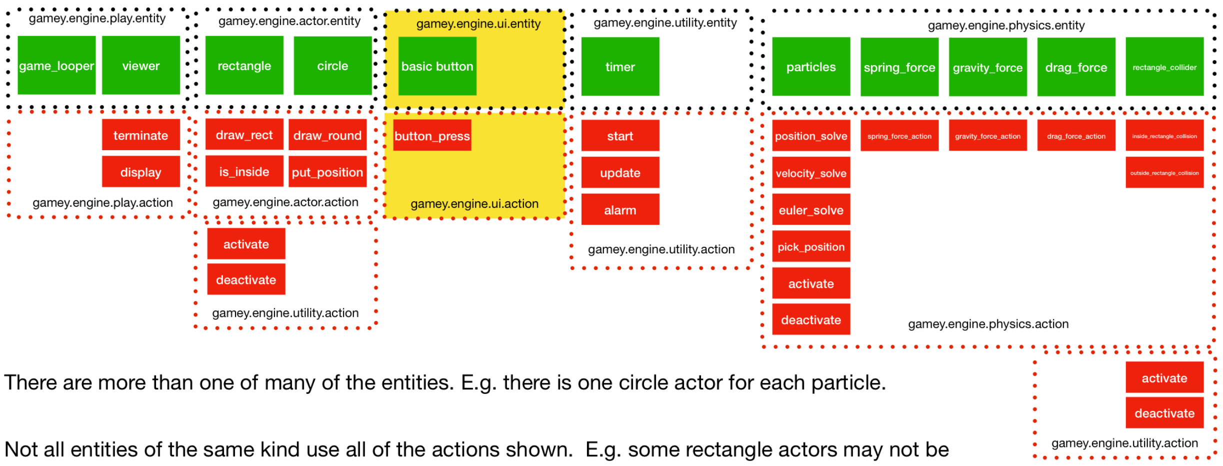


At any time, pressing the escape key or the quit button on the window should end the game and python should terminate.

The game code itself should be contained in the “python” directory, called “cannon_fodder.py”, as a python script that imports and uses the various elements of the game engine to initialize the game, entities, actions, and then runs the game loop.

This game requires the following entities and actions in the following engines:

cannon_fodder



- There are more than one of many of the entities. E.g. there is one circle actor for each particle.
- Not all entities of the same kind use all of the actions shown. E.g. some rectangle actors may not be drawn, and some rectangle actors may not require the the is_inside action.
- The physics engine activate and deactivate actions can be used to turn on/off selected particles while during the force/acceleration calculation. This can allow control of which forces are experienced by each particle.
- The entity and action in yellow boxes are for the 6160 part of the assignment only.

NOTE: THIS GAME DOES NOT REQUIRE CUSTOM ENTITIES OR ACTIONS THAT ARE NOT CONTAINED IN THE ENGINE. ALL ENTITIES AND ACTIONS SHOULD BE IMPORTED FROM THE GAME ENGINE.

STUDENTS IN 6160:

If you are in 6160, there are additional implementation elements. If you are in 4160, you may implement these also because you may find them useful. But 4160 grades will not be impacted by this additional detail.

In addition to the game as described above, you must also implement a button that is the size of the window, along with a button press action. Each time the button is pressed, the particles are given a boost of energy in the form of a drag force with negative drag constant. But this force is applied just once and then stops. When the button is pushed again, this “negative” drag force is applied to particles again, then stops. This requires the entity and action in yellow in the diagram, and also other types of entities and actions already in use in the game.