Project 7

Added by Christopher Hoder, last edited by Christopher Hoder on Nov 06, 2011

CS231 Project 7 Emergent Behavior
For this project we chose to simulate something that we came up with and designed. For my project I chose to simulate a group of Attackers raiding a Castle which is defended by Defenders. This simulation is based off of the Tower defense game as well as the stick castle defense game, http://www.castledefense.org/game/5/Stick_Castle_Defense.html. Although to make the coding assignment some of the rules and interactions were simplified. The layout of the game is simple. We have a Castle wall on the right side of the Landscape. This castle has a given health. An assorted amount of defenders are placed on the right and shoot bullets at oncoming attackers. An infinitely large amount of attackers feed in from the left with varying speed and health. The attackers travel horizontally and when they get within 1 unit of the castle, they begin to attack the castle.

The Narrative

The Attacker

- If the attacker's health is 0 it is dead and will be removed from the landscape

- The attacker will move horizontally toward the Castle at a given speed (saved as a data field and different for each Attacker).

    - If the attacker is within 2 units of the castle, the Attacker will attack the castle (with a preset amount of damage per iteration, random between Attackers)

The Army

- Will find the closest Attacker to it's position and fire 1 bullet in that direction

- With a small probability the Army object will reposition himself to a new, random location behind the castle

The Castle

- Will check to see if it has health. If it's health is 0 the castle has fallen and the simulation is over

- Will rebuild itself it it's health isn't full

The Bullet

- Will look to see if any Attackers are in it's path (between it's current position and position it will be in after moving at a given speed in the direction of motion

    - If there are Attackers in the Bullet's path, the bullet will hit the closest one

    - Otherwise the bullet will travel forward at it's given speed. If the bullet travel's off the landscape however, it should be removed from the simulation

Attacker Spawner

- This class will spawn a given number of new Attackers on the landscape with each iteration. The health, position, damage ability and speed can be all chosen at random within a reasonable range.

Implementation

The Attacker: The attacker will store his health, position, speed and amount of damage he can inflict. The updateState method for the attacker will only be concerned with advancing the Attacker closer to the castle or attacking the castle. The bullet class will deal with hitting an Attacker. The draw method of the Attacker will draw Attacker on the landscape such that the transparency of the color depends on each Attacker's health relative to the Max Health of any attacker.

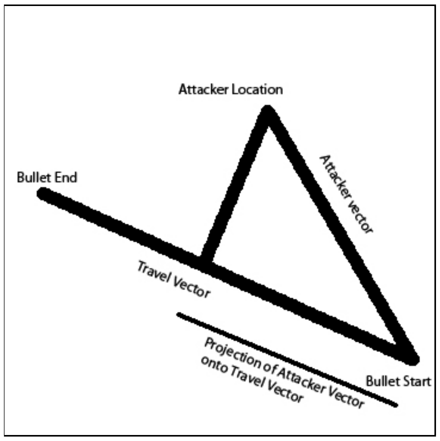
The Army: An army object will, in the updateState method find the closest Attacker to it. This was done doing a simple linear search because due to the various locations of Army objects, and Attacker objects it would have been very difficult to implement an efficient sorting or advanced searching algorithm that would easily work fro all of the army objects to find the closet Attacker. Once the Army object has the closest Attacker it will fire a bullet (create a new Bullet object) which is headed in the direction of the target Attacker. However since the Attacker is moving forward it will never hit that specifc Attacker but may hit a different one in the path. We are assuming that we have a very bad Army that does not know how to shoot ahead of a target so that it moves into the path of the bullet. Additionally, each Army object will reposition itself randomly behind the Castle. This repositioning will only occur however with really small probability.

The Castle: This object does little besides rebuilding by a set amount with each iteration. The Attackers will attack the Castle in their own updateState. the updateState method however of the Castle returns false when the entire simulation has completed, and the castle has fallen. The Castle's draw method will also draw the castle (a long gray rectangle) whose transparency corresponds to the health of the Castle relative to it's max Health. Additionally, it will display the health of the Castle on the right of the display (but off of the Landscape).

The Bullet: This class has the most complicated updateState method. First the bullet is fired at a given point on the landscape, but also must be able to travel beyond that point. For this reason, the object will store the directional vector and not just the target point. In the updateState method, the bullet must determine whether or not it hits an Attacker in it's path. The way that I completed this code was to use vector projections and pythagorean's theorem. You do this by finding the vector from the bullet's start point to it's finish point. Call this the travel vector. Then you find the vector from the bullet's start point to each Attacker. Call this vector the Attacker vector. Then the code computes the projection of the Attacker vector onto the travel vector. The formula for this is :

$$\boxed{\frac{\overrightarrow{Travel} \bullet \overrightarrow{Attacker}}{\left\|Travel\right\|}}$$

Then the distance the Attacker is to the path travelled by the bullet can be found using the standard pythagorean formula: r^2 = x^2 + y^2. An example is shown below of what the calculations are telling us. From this image you can easily see the triangle that you create when you figure out the projection of the Attacker vector onto the Travel vector.

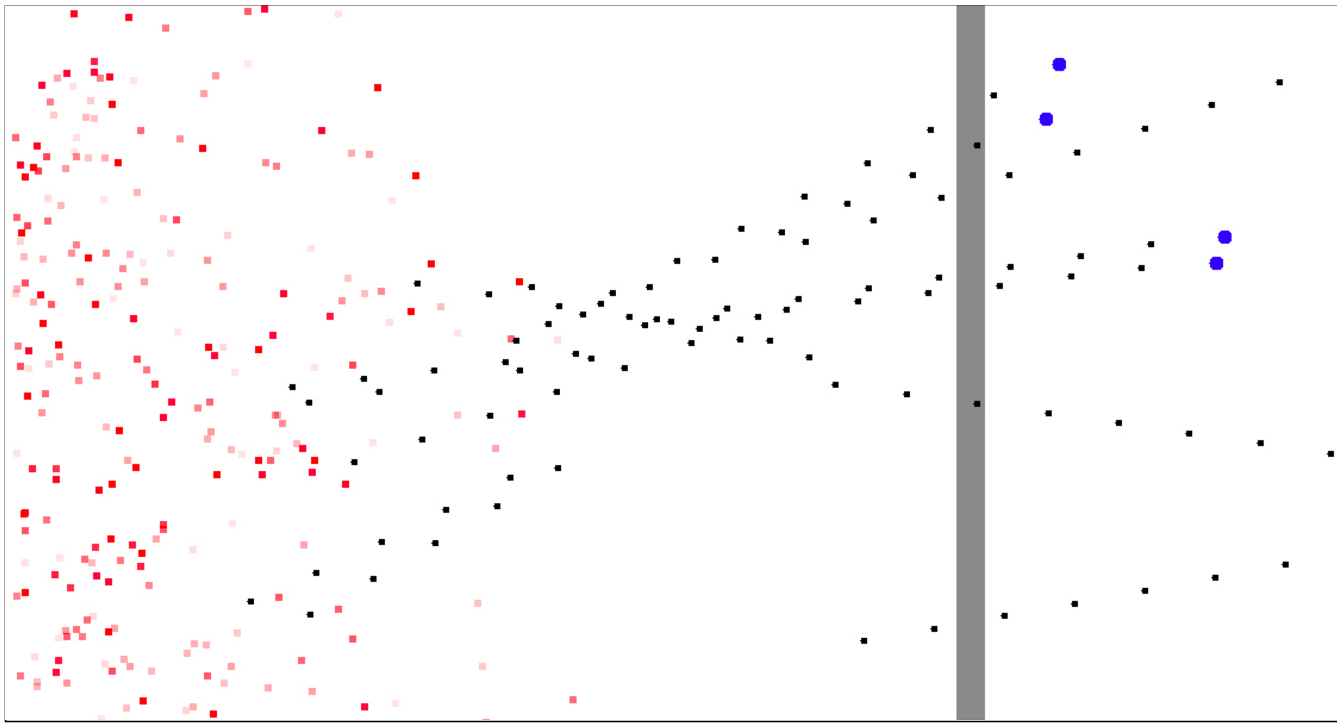


If the bullet hit's an Attacker it must cause damage on the Attacker and remove itself from the Landscape. Likewise if the Bullet goes off the end of the Landscape it must also remove itself from the Landscape.

Extensions

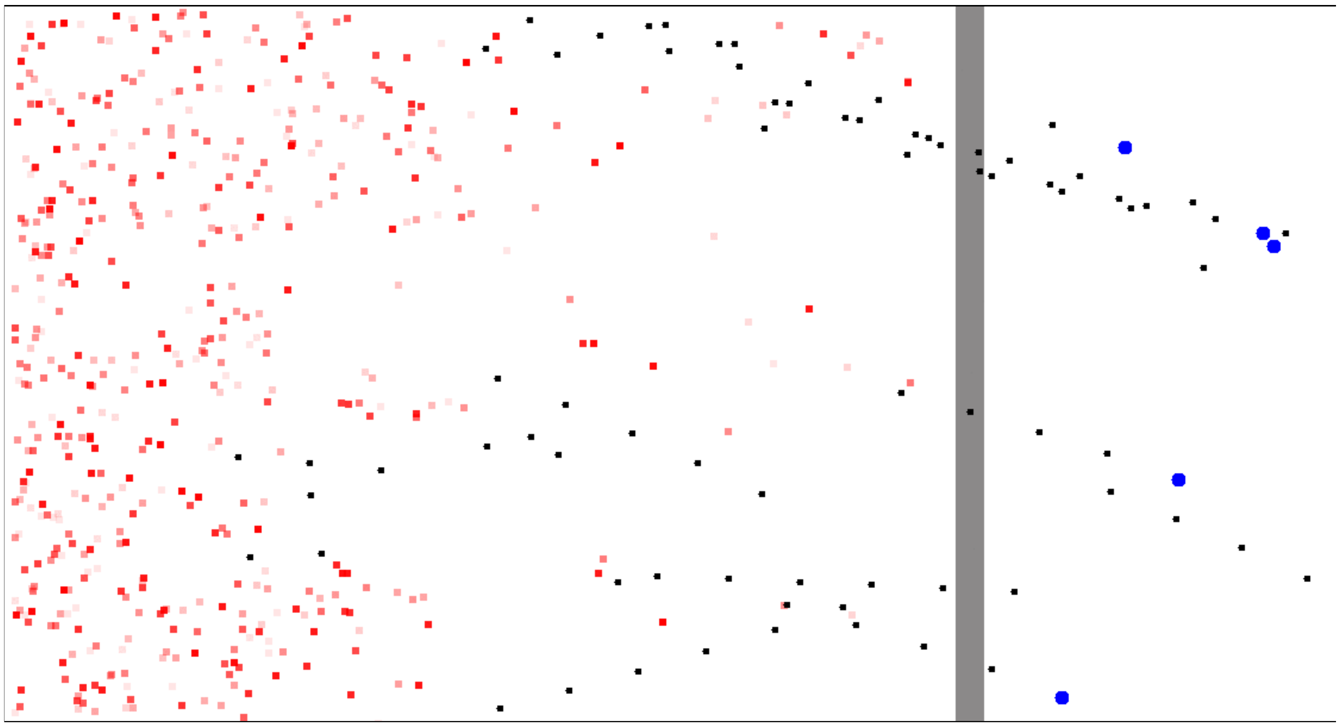
As some extensions for this project. I worked to tighten up my code by adding try-catch code where ever there was an exception, as well as changed all of my lists to include generic type calls so that there was less type casting needed. Additionally, not knowing a reasonable value for any of my parameters in the simulation I designed the code as such to include a lot of static Maximums for each class. These static variables are declared at the top of each class and changing them allows me to change variables in my code, such as the Max health of various classes without having to go into the code and determine everywhere it is called. Additionally, I rearranged how the display was being created so that it has a boarder on the right side that is not part of the landscape but can display relevant information about the simulation. An image below demonstrates this, there is a black line dividing this section and the edge of the landscape.

After completing the basic goals of my simulation I wanted to make it more interesting so I changed things to introduce a bit of variety and randomness that is often seen in battle. I made most data field in the simulation get assigned randomly so that every Attacker was different. They travel at different speeds, have different amounts of health, appear in different places and can cause different amounts of damage. Additionally, a random (but bounded) number of Attackers is spawned with each iteration, not a set number. Furthermore, randomly (at least once every 300 iterations) there will be a surge of Attackers spawned for 10 iterations. The number of iterations until the next surge is given to the viewer on the right of the display.

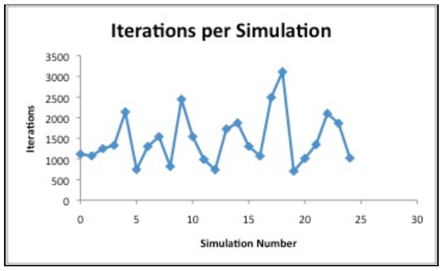


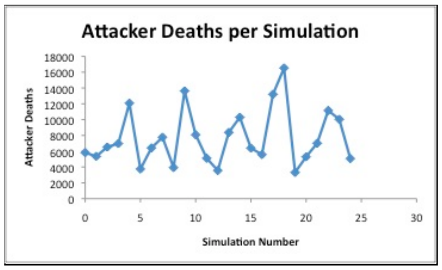
Additionally, since this simulation has a plausible end, with the castle falling. I added some code that will allow us to simulation several of these attack simulations (with the same beginning parameters) and save the data to a text file for each simulation. We can then compare the simulations. The data that will be saved at the end of each simulation is the number of iteration it took for the castle to fall as well as the total number of attackers killed before the Castle fell. The syntax for calling these options from the command line can be read from the comments in the BattleSimulation syntax.

Below is gif of the Battle Simulation running under normal default parameters, You can see the data display on the right about where we are in the simulation. These simulations under these parameters can run for a long time before the castle falls. The random troop surges seem to make a considerable different in whether or not the castle falls. If the surges happen fairly spaced out then the Army is able to kill off the additional troops but if several surges happen frequently together, the Army becomes overwhelmed and cannot recover in time. This particular simulation ran for over 1700 iterations before being overwhelmed by Attackers.



Using the option to save simulation information, I ran the simulation 25 times and saved the data for each simulation in data.txt (included in the handin). All settings were default except that I set there to be at most 10 spawns per iteration to make the simulations shorter. I limited trials to 25 because of time restrictions. Each simulation can take a long time. I then loaded the simulations into Excel to produce the graphs below.





From these graphs we can conclude a few things. First that the number of iterations and the number of Attackers killed is highly correlated (no surprise here). Additionally we see a wild variation in the number of iterations required for the castle to fall. Using this data in conjunction with watching my simulation run, it is obvious that there are two major components in determining if the Castle will fall. First is that a few surges need to come together within very few iterations. This balloons the number of Attackers on the landscape. Second, the speed of these new Attackers must be fairly fast (close to the Max Speed). We need the Attackers to be fast so that they have moved out of the path of any bullets fired at them (remember that the Army isn't smart and aims only at where they are and not where they are going).

In conclusion this project helped to show me how to go about developing my own simulation and figure out what aspects of a senario I wanted to include and which ones were best to ignore. Additionally, it taught me about the necessary thought process needed to design what each class's method's and fields are going to be before I begin coding. I used this project to help me learn more about some of the different topics we have been discussing in class recently including exceptions and generics. Both of which I tried to use when possible in my code.

Labels

cs231f11project7

Printed by Atlassian Confluence 3.3.1, the Enterprise Wiki.