Criterion C: Development

Techniques Used

a. Graphical Interface

b. Variables

c. Lists

d. Methods

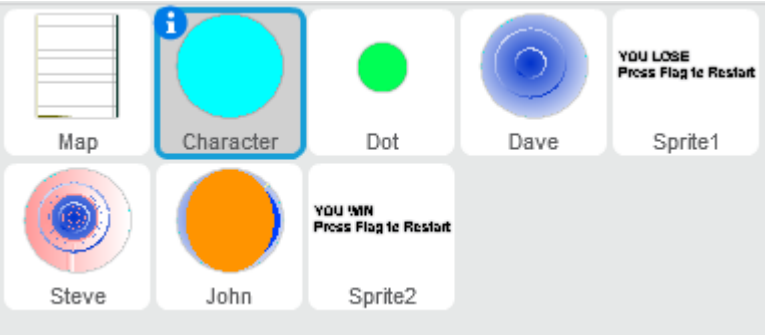
e. Sensing algorithms

f. AI algorithms

g. Sorting algorithms

Graphical User Interface

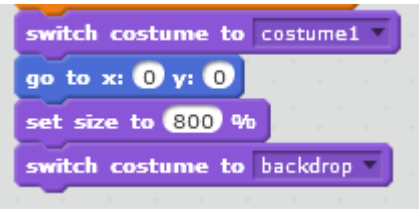
Scratch has a built-in graphical user interface template involving a stage and sprites.



Sprite1 and Sprite2 are the losing and winning text that appears when one of the endgame condition are met. They will hide themselves when the game starts.

The Character, which the player controls, is fixed at the center of the stage. The AI's identifiers are Dave, Stave, and John.

The Map sprite contains a costume that is an image of a grid.

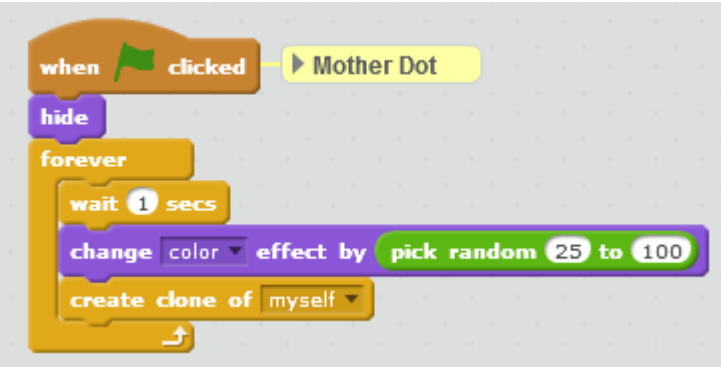


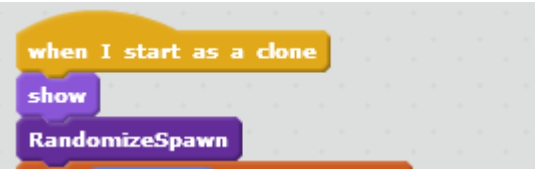
The above sequence allows the costume to be zoomed in 8 times, forming a map that has parts existing outside of the stage.

The following code sequence decides how the map moves. The y-coordinates and x-coordinates of the map change depending on where the player's cursor is pointing, creating the illusion that the Character is moving on the map with a fixed camera and point of view.

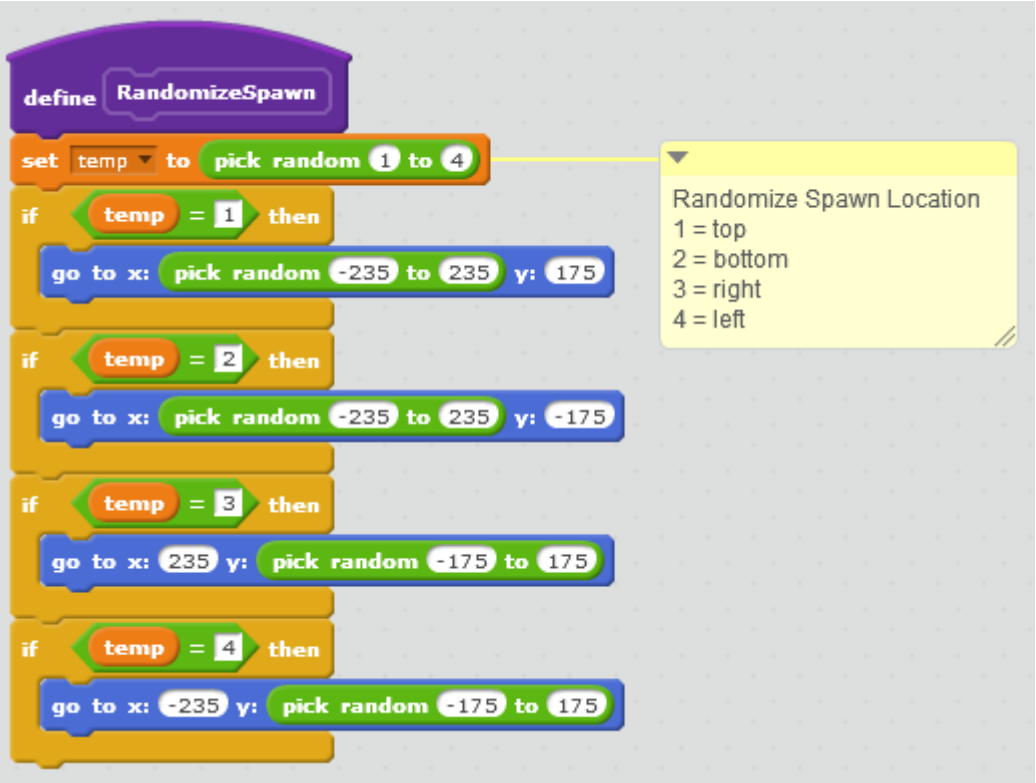
The above code is also copied to all other sprites except the Character. This ensures that all other objects are moving relative to the Character.

The below code shows how dots are generated.





A dot is generated every second and its color effect is also randomized. The cloning block is used efficiently here as multiple instances of dots appear throughout the game. When they appear, their location is randomly generated with the RandomizeSpawn function, shown below.



The function is a randomizing algorithm that first randomly chooses one of the four edges, and then randomly chooses its other axis position.

Variables

Local Variables:

myIndex – integers used to hold the index of each dot when they spawn

Global Variables

PlayerScore, CharacterSize – integers that holds the values of the player Character's score (number of dots eaten) and size of the Character

DaveScore, DaveSize, SteveScore, DaveSize, JohnScore, JohnSize – integers that hold the values of the AI's scores and size

PlayerSpeed, DaveSpeed, SteveSpeed, JohnSpeed – double type values that hold the player and AI character's speed factors

AtXBorders, AtYBorders – booleans that hold the status of whether **Character** is touching the left/right or top/bottom borders, respectively. Not required for **AI Characters** because **AI** algorithms will not allow them to move towards a border (since no dots will spawn beyond map and **Character** cannot move beyond map).

These variables are set to global because Scratch has limited programming capacity. If they were local, a method would have to be defined for each object to get the values. In Scratch 2,

they could be done through broadcasting, but would result in less elegant code structure. Thus, I decided to set global variables with identifiers to create a simpler, elegant, and understandable code structure.

Lists

Lists are similar to arrays in other common programming languages. All lists are displayable on the stage.

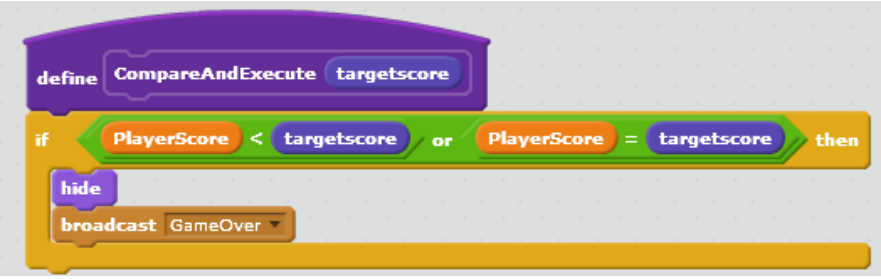
Rank – holds the Strings “Player”, “Dave”, “Steve”, and “John”

Score – holds the scores of each character

dotxvalue, dotyvalue – holds the x and y coordinate of dots respectively, added whenever when a dot is spawned, works alongside each dot’s myIndex. Used for AI algorithms.

Methods

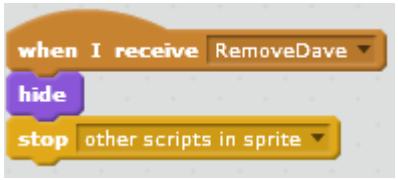
In Scratch, methods can be called through broadcasting. Broadcasting to call methods are also used when two characters interact with each other. The below code shows how the Character interacts with the AI characters. Once they are in contact, their size will be compared with the CompareAndExecute function that takes in the current score of the target.



The function compares the scores of the two characters and calls the method GameOver.

GameOver is received by Sprite1, the losing message. The game ends and displays losing message.

If CompareAndExecute does broadcast GameOver, the game continues and means that the **Character** ate the **AI character**. So the size, score and speeds are updated.



RemoveDave is received by Dave, and Dave’s scripts are stopped. The same applies to RemoveSteve and RemoveJohn.

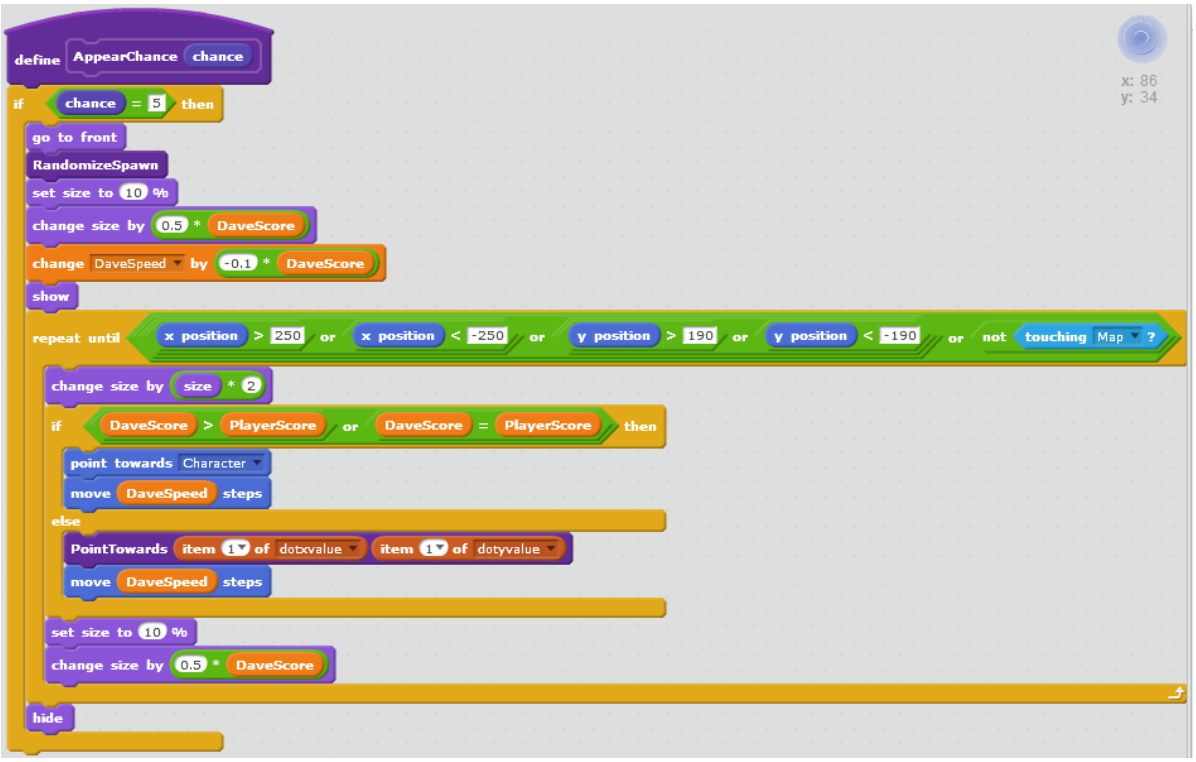
Sensing Algorithms

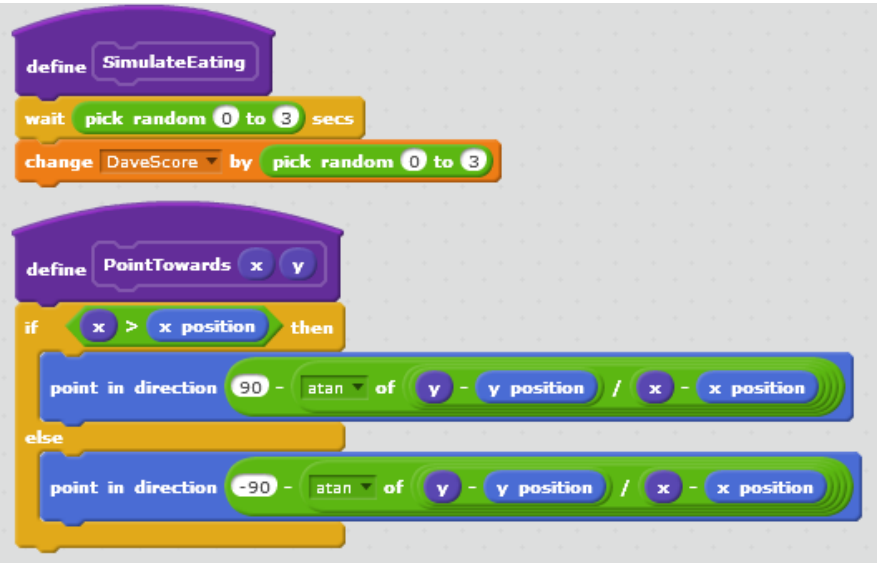
Sensing algorithms are used to set AtYBorders and AtXBorders value.

The **Character** remains in the map for the entirety of the game. Each of the four sides of the map is marked by a specific color, unique throughout. Through sensing the different colors, the algorithm can determine which border the **Character** is on.

AI Algorithms

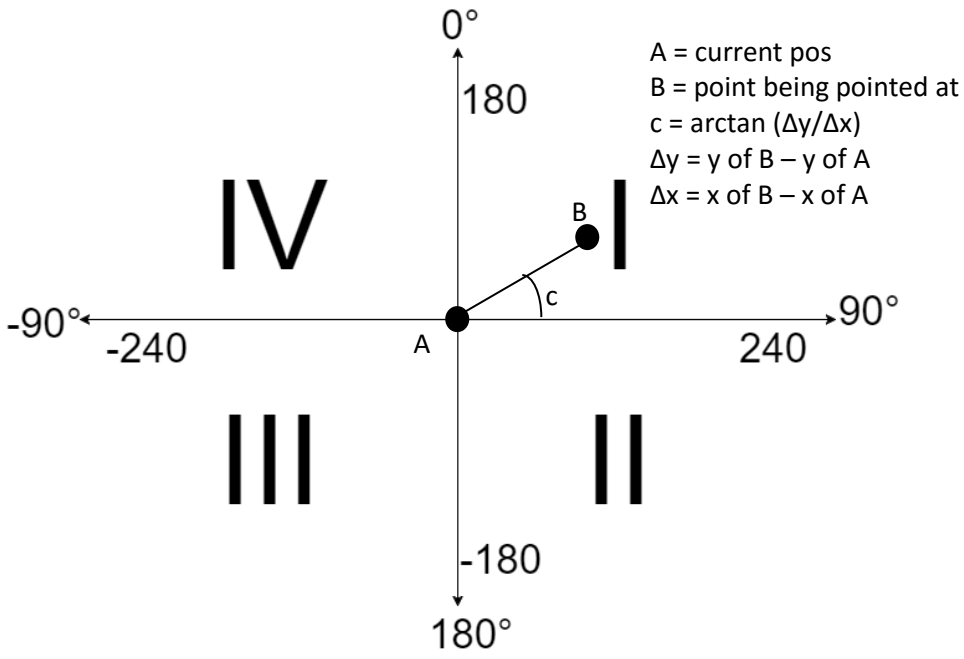
There are a total of three AIs that all contain the following key functions.





When the game begins, each AI will loop through AppearChance and SimulateEating. AppearChance follows a randomizing algorithm that determines if the AI should appear in the stage at the moment. **AI Characters** will forever select a random integer, and if it equates value 5 (or any other number set in AppearChance), the **AI Character** will set its size and speed based on its score and appear. Their spawn location reuses the RandomizeSpawn function.

Once inside the stage, the score of the AI and the player is compared. If larger, the AI will move towards **Character**, otherwise it will move towards a dot. PointTowards function is a mathematical algorithm to determine the direction by taking in two x, y values. To get a better understanding, refer to the layout of a Scratch 2 stage below.



If the dot belongs in **I,** $\Delta y/\Delta x > 0$, $0 \leq c \leq 90$, so direction is $90° - c$.

Case **II,** $\Delta y/\Delta x < 0$, $-90 \leq c \leq 0$, so direction is $-90° - c$.

Case **III,** $\Delta y/\Delta x > 0$, $0 \leq c \leq 90$, so direction is $-90° - c$.

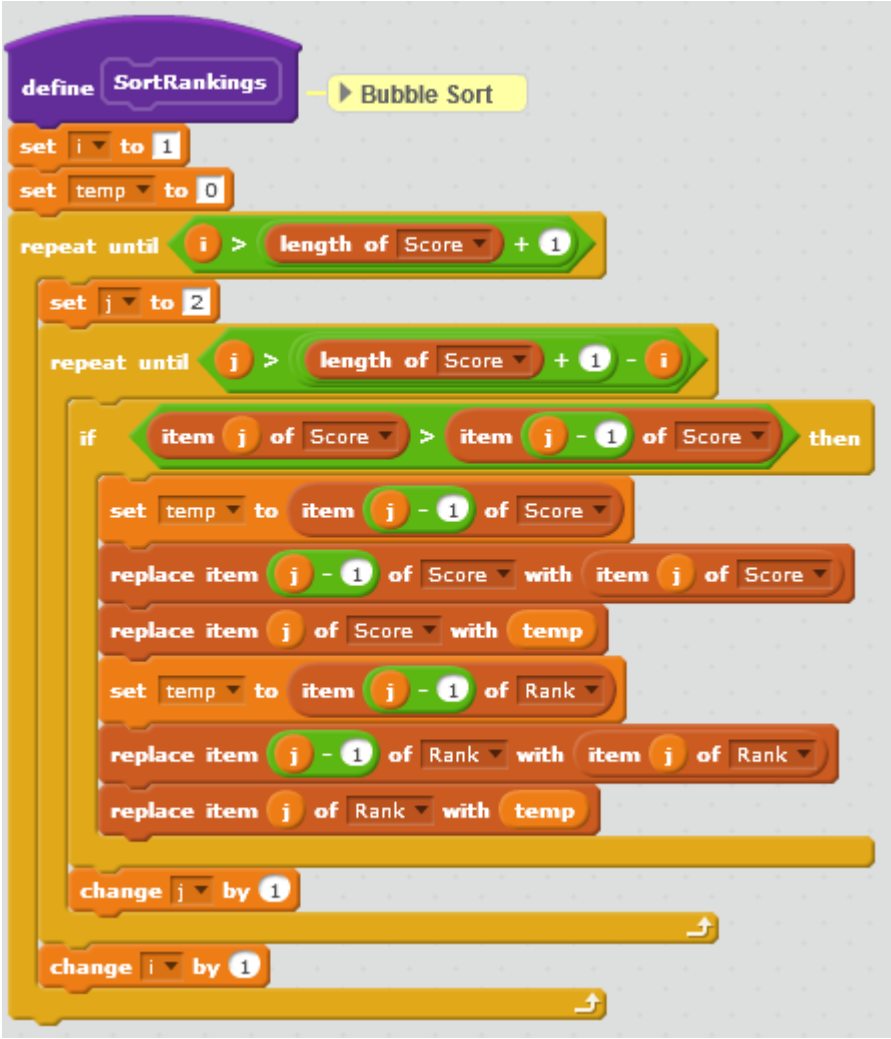
Case **IV,** $\Delta y/\Delta x < 0$, $-90 \leq c \leq 0$, so direction is $90° - c$.

As seen, for **I** and **IV**, the direction is $90° - c$, meaning AI xpos > dot xpos. For **II** and **III**, the direction is $-90° - c$, meaning AI xpos < dot xpos. Thus an if statement is used in PointTowards to separate the two cases to determine the right direction.

If **AI Characters** touch the edge of the stage, they will hide themselves and begin SimulateEating, a randomizing algorithm to increment the scores of the characters while AIs are offstage.

Sorting Algorithms

Score list is constantly sorted using BubbleSort algorithm.



Additionally, the Rank list that contains the names of all characters are sorted as well respective to the score each name refers to. This creates a scoreboard that ranks all characters in order and assists the player in determining their actions.

Word Count 1055