Zach Hopman

Bo Zhang

**Chase**

**Abstract:**

For our final project in Intro to Programming, we are making a game similar to “Tilt to Live”: <http://onemanleft.com/games/tilttolive/>. The main goal of the game is to race away from the enemy, red dots, when they chase the your “character”, an arrow, while at the same time picking up power ups to destroy the dots, allowing sustained game play. Instead of tilting a handheld device as shown in the link, we will implement the side arrow keys to rotate the arrow, and the forwards arrow key to move it in the direction it’s facing. Our primary goals are to detect collision among the arrow, enemy pieces, and field boundary, implement a chasing mechanism, allow the destruction of dots with offensive power ups, and incorporate power ups that are more defensive oriented such as a shield or increased movement speed.

**Programming Skills:**

* Variables: keeping track of the number of dots killed, rotation degree, power up obtained, dot and player (x,y) positions-perhaps with a class-, and positions of sprites of attacks from power ups.
* Conditionals: checking for collision, dot destruction, and user input(arrow keys).
* Loops: game loop to keep everything running and buffer checking for user input, initializing the starting positions of multitudes of pieces.
* Functions: the effects of each power up, difficulty, collision detection, kill count and wave value to graphical display conversion-since we cannot “cout” numerical values on GUIs.
* Arrays: used for storing the locations of player and enemy (x,y) positions.

**Data Structure:**

To run the game, we will have a giant loop that constantly checks if you have moved your piece, lost through collision with an enemy dot, gained a power up, or killed an enemy piece. Inside the loop will be conditionals for the boundaries of the board and for the different power up locations to determine if they are touched. Kill count will be incremented by destroying enemy pieces (+1 for every dot). We have decided to keep track of the position of enemy pieces and power ups with structures (So we can access them like enemy.x or enemy.y).

As for the functions:

* **Power up**: Multiple versions of these, just with different sprite rendering and effects. For example, a function called Nuke would destroy all the dots and replace their images with that of a mushroom cloud. Also keeps track of deleting and making new power ups when old ones are used so that there will always be 3 -4 power ups on the screen. Perhaps a feature that allows better power ups to spawn in later.
* **Conversion**: Takes a numerical parameter and displays to the screen a set of individual PNGs of numbers that correspond to the parameter.
* **Difficulty**: Man, we loved coming up with this one. There will be a variable that keeps track of what wave the player is on, starting from one and incrementing every time a set of spawned dots are killed. As the waves progress, the radius and number of dots will increase/decrease. How so you ask? Radius: 1.5^((wave/2)sin(wave/2)) + 10 <-- Slant Asymptote! Number of dots: Keep at 10 unless radius is less than 10, then the equation is 10 + (8/45) \* wave, truncated of course.
* **Chasing**: Use a distance formula to determine the distance between individual dots and player. Then use trigonometry to determine the angle the dot needs to travel to reach the player and make it move in that direction.
* **Collision**: Use the distance formula to see if two boundaries have overlapped. If so, return a Boolean value as true to destroy that object.
* **Formations**: This is only if we have spare time. As the waves of the game progresses, the dots will no longer spawn in random positions but rather militaristic formations and attack the player in more organized fashions.

**Deadlines:**  see spread sheet.