Team Name: WaffledOctopie
Terry Guan, Queenie Xiang, Fabiola Radosav
Pd.3

ANTIKYTHERA

Description:

Antikythera is an original game idea where a character is placed on a big hollow circle. The character constantly runs along either the outer or inner edge of the circle and has the option to switch between these two edges of the circle. Along both edges random enemies are spawned and the character has to avoid these enemies by switching which edge it is on.

Features:

- **Structure**
  - (Processing files)
  - Driver File: Antikythera
  - Super Class : Character
  - Sub Class: Enemy
  - Sub Class: Player
- **Gameplay Process**
  - Objective: obtain the highest score possible
  - The score is based off of the amount of time the player survives and the upgrades the player is able to collect
  - The game will continue until the player perishes by coming in contact with a monster
- **Player character**
  - Player is always in motion around a circle at a constant velocity.
  - Player starts by spawning on top of the outside edge of circle (0 degrees position).
  - Player has the ability to switch between outer and inner edge of circle by hitting the spacebar.
    - This is accomplished by using processing's method ***KeyPressed()*** and variable ***key.***
- **Power-ups**
  - Randomly generated both in and out of the circle and/or bought with points. The player can store up to 3 upgrades but can only use the most recent upgrade it collected.
  - The upgrades will be displayed on the processing screen allowing the user to see what's next.
  - Limitation: player will not be able to remove power-ups they do not want.
  - Power-ups are used by having the user press a not-yet-assigned key
    - The stack will be updated
  - Power-ups will be created as different shapes
    - Processing methods that are necessary:
      - ***rect()***
      - ***createShape()***

        - ***quad()***
        - ***triangle()***
- "**Gains and loss" system**
    - Player has the option to buy an upgrade with the points they have gained so far. While player may have to sacrifice his or her score for an upgrade, he or she can ultimately improve his or her chance of getting a higher score.
    - Player will be able to pause the game for ten seconds to buy an upgrade
- **Enemies**
    - There will be different types of enemies.
    - Different categories of enemies will have different priorities.
    - The enemies will be spawned depending on the priority, at a random location.
    - The duration between every new birth of an enemy will be kept track of by a timer.
- **Mechanics of Time Tracking**
    - Will be used in:
        - Lapse between enemy spawning
        - Pausing the game (stopping the score updates) so player can purchase upgrades
        - Updating score
    - Will use ***seconds()*** and ***minutes()*** methods of processing to record the various times needed
- **IsTouching()**
    - An isTouching() method will be implemented to determine if the enemy and the player are touching
    - If the player and enemy touches, then the player will die, and the game will terminate
- **Polar Coordinates**
    - Polar coordinates are used to:
        - Move the enemies and player in a circle
        - Draw the enemy and player
            - I.e. spawning new enemies at random locations with random polar coordinates
- ***Optional Feature***
    - There will be a maze that's generated inside of the circle.
    - The character would have to navigate the maze every time it wants to enter and exit the inside circle
        - There can be a mode where game turns into temporary maze
        - The user can have the opportunity to gain another powerup if he/she successfully completes the maze under a certain amount of time

Most Basic Implementation:

- The game will be properly set up and drawn
    - There will be a circle which the player and enemy will move around
    - One player and one type of enemy will be spawned.
    - The character must avoid enemies:
        - By having the ability to switch between the inside and outside of the circle.

Basic Class Concepts:

- **Heap Priority Queue**
    - Will be used to hold enemies that are awaiting to be spawned
    - Enemies are appended based on the level difficulty global variable in the driver file:
        - At the beginning levels, easier monsters will have a higher percentage of being appended to the queue
        - As the levels get harder, harder monsters will have a higher percentage of being appended to the queue
        - This queue will be sorted with **heap sort** so that the monster at the last index (size-1) will always have the highest priority
            - Diagram:
                - Head (index 0) Tail (index size-1)

Lowest Priority	→	→	Highest Priority

        - After arbitrary amount of seconds (randomly determined), the monster at last index will be “popped off” and spawned
- **Stack**
    - The upgrades will be kept in a stack. Since the player can only use the most recent collected one it would make sense to have a stack. The stack would pop the upgrades one by one.