INFO1113 Assignment 2 Report

SID: 500214288

Object-oriented design decisions

Ghosts

1. The only difference between the 4 types of ghosts is the way they get the target. So we only need to write one complete class of ghosts.
2. For me, I choose "Chaser" as the superclass, then the other three ghosts (Whim, Ignorant, Ambusher) inherit all instance fields and methods of this superclass but override the method "updateTarget" to achieve unique behaviour.

Fruit

1. Things that can be eaten by waka also have many of the same attributes, including fruit, super fruit and soda. The same as ghosts, we choose Fruit as the superclass whose subclass are SuperFruit and Soda.

   Waka needs to know what kind of food it eats to determine the mode of itself (frightened) and all ghosts (frightened/invisible), so we need methods to judge the type of food.

   ```
   // Consider this is part of class Fruit
   public boolean isFruit(){
       return true;
   }

   public boolean isSuperFruit(){
       return false;
   }

   public boolean isSoda(){
       return false;
   }



   // Consider this is part of class SuperFruit
   @Override
   public boolean isFruit(){
       return false;
   }
   ```

```
@Override
public boolean isSuperFruit(){
    return true;
}



// Consider this is part of class Soda
@Override
public boolean isFruit(){
    return false;
}

@Override
public boolean isSoda(){
    return true;
}
```

Extension

- Collectable soda-can that frightens ghosts and turns them invisible for a period of time

1. Player (waka) has a method "eat" which will return the type of food the Player eats. If it is soda-can, it will return "Soda"

```
// If the thing be eaten is soda-can it will return "Soda"
// If the thing be eaten is super fruit it will return "superFruit"
// If the thing be eaten is common fruit it will return "fruit"
public String eat(List<Fruit> fruits, List<Chaser> ghosts) {}
```

2. According to the return value, we draw ghosts or not.

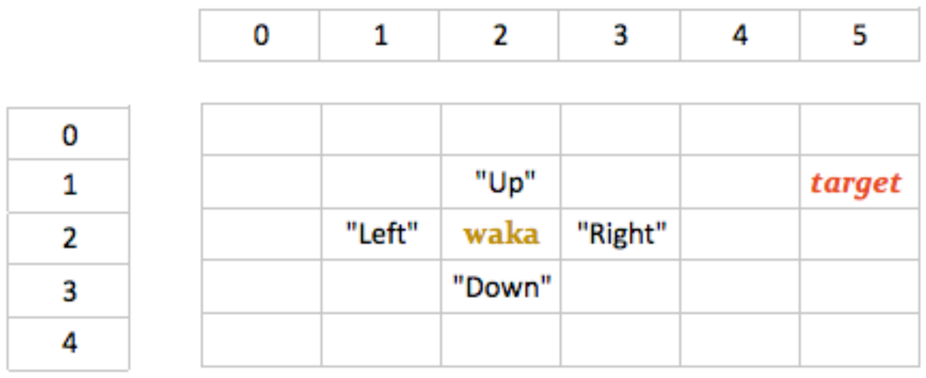
```
public static boolean drawGhosts(PApplet app, ..., List<Chaser> allGhosts,
Player player){

  String eatenThing = player.eat(fruits, allGhost)
  // then we use the return value to draw all ghosts or not(invisible)

}
```

How ghosts move ?

1. Determine the priority of the four directions based on distance



Sort the distances from the four grids around waka to the target

```
distance = sqrt((x1 - x2)^2 + (y1 - y2)^2)

int[]     distances  = [      √5,    √9,     √13,     √17]
String[] priorities = ["Right", "Up", "Down", "Left"]
```

2. What are the conditions for going in a certain direction ?
   - The next grid in this direction is not a wall
   - Can't go in the opposite direction of the current direction
     - The ghost will never go back at the intersection, causing it to linger.

```
 public boolean canGo(String direction, MyMap map){
      // When satisfying both conditions return ture, otherwise return false.
    }
```

3. Let's set direction.

```
public void setPriorityDirection(List<String> priorities, MyMap map){
      for (String priority : priorities){
        if (canGo(priority, map)){
          setDirection(priority);
          break;
        }
      }
}
```

About Time

Mode Length

```
modeLengths: [2,  20,   2,   30,   2,  100]
mode:        [s,   c,   s,    c,    s,    c]
// Elements with even index are Scatter modes, elements with odd index is
Chase modes
```

Link time and mode through indexs.

```
modeLengths: [2,  20,   2,   30,   2,  100]
mode:        [s,   c,   s,    c,    s,    c]
prefixSum: [0,   2,   22,   24,   54,   56,   156]

The value of (currentTime % prefixSum[prefixSum.length - 1]) must be
between two adjacent elements in the prefix list.

For exmaple: current time (second) : 200 % 156 = 44  (24 < 44 < 54)

The index of 24 is 3, 3 % 2 != 0, so it is in Chase mode.
```