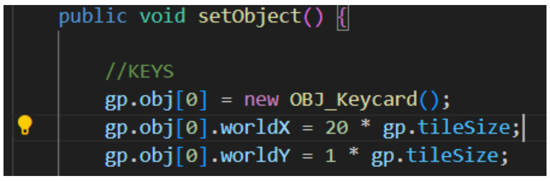
Alex's code review on Ran's code

Code duplication in AssetSetter:

The AssetSetter class manually sets SuperObjects using this pattern:



```
public void setObject() {

    //KEYS
    gp.obj[0] = new OBJ_Keycard();
    gp.obj[0].worldX = 20 * gp.tileSize;
    gp.obj[0].worldY = 1 * gp.tileSize;
```

Not only do indices need to be manually specified, but positions always are multiplied by gp.tileSize. A fix is to create a method that automatically sets objects based on an array of positions and a class constructor. I also noticed that the original indices skipped some indices of the object array due to human error when entering the indices, which meant that some test classes needed to be updated after refactoring to account for the proper sequential object ordering.

Commits: 52d6cd094dd3b50a919486b99e08d851af7cd5d2, 97b8bd3c34551d413cd3be07530d4355b208f464, 7c000297e95a04cec6558cdfdcd777e1ae06d139, 1cf885de362a111bea788e1c38d2ec063f62ae06

Unnecessary variables - solidAreaDefaultX/Y in SuperObject class:

SuperObject.solidAreaDefaultX is associated with the SuperObject class but is only used once in the CollisionChecker.checkObject() method. Also, the .checkObject() method needs to compute the rectangle indicating the object's collision box every time that the method is called even though the rectangle always remains at the same position. A better approach is to create a .setObjectPosition() method for SuperObjects that both sets their X and Y positions and simultaneously computes a collision box. This required changing all SuperObject position assignments to use this new method.

Commit: d68869b16629fd9a28b3171d26a9b8658b8773d4

Overly long method - UI.draw():

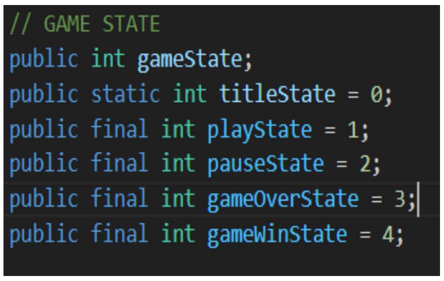
UI.draw() handles title, pause, and game over states by calling a private method, however it handles the playing state and win state entirely inside the method, making it a bloated method. The solution here is straightforward - simply move the play and win state drawing into their own separate methods.

Commit: 73727a21bae4f55cfc73700bcf3f95aa414a150b

Unjustified use of primitives to control game state in GamePanel:

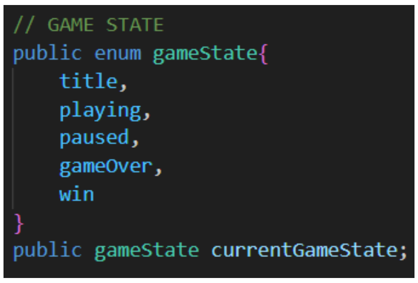
Game state in the GamePanel object is controlled with an integer and can be set to various final integer values. This is less safe and readable than using an enum. This was fixed by storing the game state as an enum rather than an integer.

Commit: 05fbf1deb9b8cc68220b3ecb8a5297000b72aa6c



```
// GAME STATE
public int gameState;
public static int titleState = 0;
public final int playState = 1;
public final int pauseState = 2;
public final int gameOverState = 3;
public final int gameWinState = 4;
```

becomes



```
// GAME STATE
public enum gameState{
    title,
    playing,
    paused,
    gameOver,
    win
}
public gameState currentGameState;
```

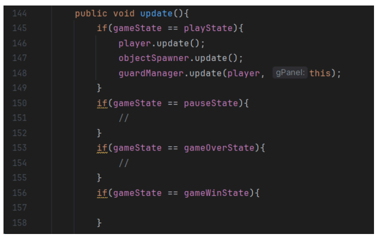
<u>Ran’s code review on Alex’s code</u>

<u>**Classes that are too large and/or try to do too much** in the update() of ObjectSpawner class</u>

The update() method is doing too much. It tracks ticks, handles object spawning, and manages old objects, which could be broken down into smaller methods. A solution is to extract the random position generation and collision checking into a separate method.

commit: 9c442e399fdd99952b351ce6518cfed6a0318207

<u>**Dead code** in the update() function of GamePanel class</u>



```
    public void update(){
        if(gameState == playState){
            player.update();
            objectSpawner.update();
            guardManager.update(player, gPanel: this);
        }
        if(gameState == pauseState){
            //
        }
        if(gameState == gameOverState){
            //
        }
        if(gameState == gameWinState){

        }
```

These 3 if statements are doing nothing. They were intended to be implemented in GamePanel class, but later we realize it will be more convenient to implement them in the UI class, and forget to remove them after we finished the game. A solution is to delete this part.

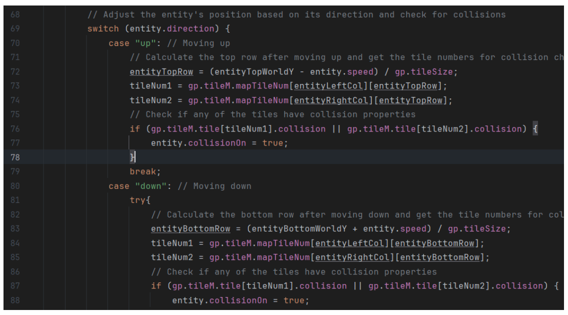
commit: 05fbf1deb9b8cc68220b3ecb8a5297000b72aa6c

<u>**Code duplication** In the moveTowardsPlayer() of Guard class</u>

It checks for collisions, setting movement directions based on the collision state, and then rechecking. The duplication arises from the recurring structure of: flipping the movement direction, setting the direction of movement with setDirection(dx, dy), toggling the collisionOn flag, and rechecking collision with gp.cChecker.checkTile(this). Each of these steps is repeated in the if-else blocks, we refactoring this repeated logic into a separate method called attemptMove(int dx, int dy)

commit: 8eee192219397b3efdcb82f3dad9d27bb6b90397

<u>**Code duplication** In the checkTile(Entity entity) of CollisionChecker class</u>



```
        // Adjust the entity's position based on its direction and check for collisions
        switch (entity.direction) {
            case "up": // Moving up
                // Calculate the top row after moving up and get the tile numbers for collision ch
                entityTopRow = (entityTopWorldY - entity.speed) / gp.tileSize;
                tileNum1 = gp.tileM.mapTileNum[entityLeftCol][entityTopRow];
                tileNum2 = gp.tileM.mapTileNum[entityRightCol][entityTopRow];
                // Check if any of the tiles have collision properties
                if (gp.tileM.tile[tileNum1].collision || gp.tileM.tile[tileNum2].collision) {
                    entity.collisionOn = true;
                }
                break;
            case "down": // Moving down
                try{
                    // Calculate the bottom row after moving down and get the tile numbers for col
                    entityBottomRow = (entityBottomWorldY + entity.speed) / gp.tileSize;
                    tileNum1 = gp.tileM.mapTileNum[entityLeftCol][entityBottomRow];
                    tileNum2 = gp.tileM.mapTileNum[entityRightCol][entityBottomRow];
                    // Check if any of the tiles have collision properties
                    if (gp.tileM.tile[tileNum1].collision || gp.tileM.tile[tileNum2].collision) {
                        entity.collisionOn = true;
```

The code for calculating tile numbers based on direction and checking for collision is repetitive across different cases in the switch statement. Each case block essentially does the same operations with slight change for the direction. A solution is to create a method for collision checking given two tile numbers and calculate the next row or column based on the direction.

commit: 348545abcf0eb8bb362e15b14e1b569cd7f58f56