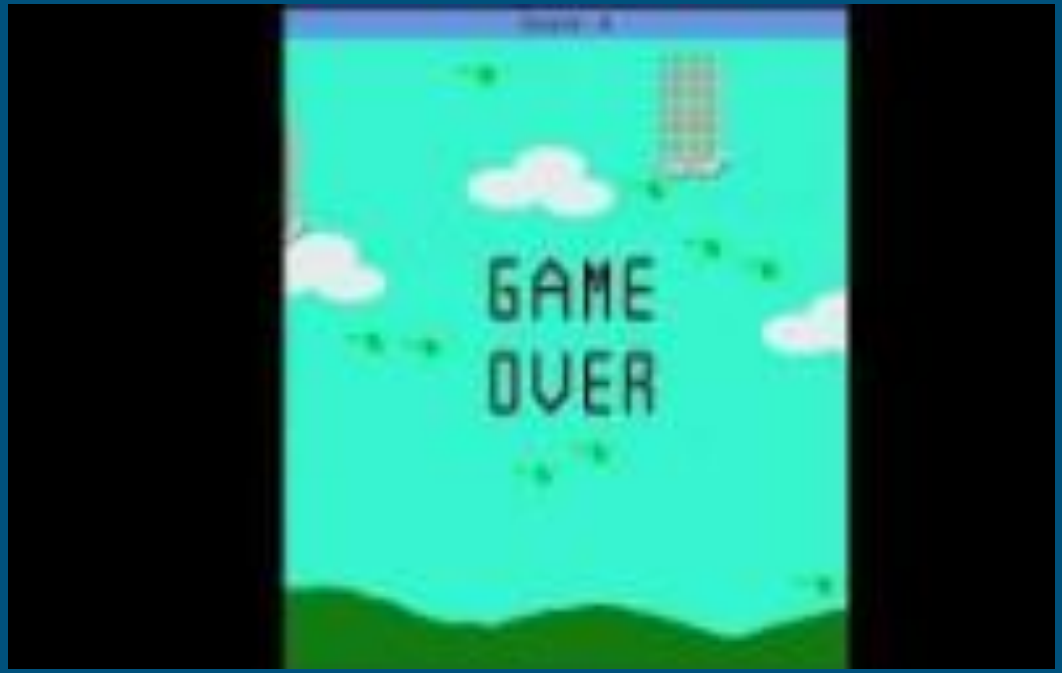
Flying Tank Simulator

By: Chelsea Salzsieder, Josh Wilger, Krissy Olson, Zachary Mineau

Demo Video

https://youtu.be/z7lv8e03nLg



Time to View Code

Sprites/Images

```
// SPRITE CLASS

    // Reusable Sprite class for adding visual objects to the game
    public static class Sprite extends ImageView {
        boolean dead = false;
//      double x_velocity = 0;
//      double y_velocity = 0;
        final String type;
        final String location;

        // Initializing the Sprite class
        Sprite(int x, int y, int size, String type, String location) {
            super(location);
            this.location = location;
            this.type = type;
            setTranslateX(x);
            setTranslateY(y);
            this.setFitHeight(size);
            this.setPreserveRatio(true);
        }

        // Sprite movement
        void moveLeft(int amount) {
            setTranslateX(getTranslateX() - amount);
        }
        void moveRight(int amount) {
            setTranslateX(getTranslateX() + amount);
        }
        void moveUp() {
            setTranslateY(getTranslateY() - 8);
        }
        void moveDown() {
            setTranslateY(getTranslateY() + 8);
        }
    }

// MAIN METHOD

    public static void main(String[] args) {
        launch(args);
    }
}
```

```
import javafx.animation.AnimationTimer;

public class FlyingTankSimulator extends Application {

// GLOBAL VARIABLES

    // Creates javafx entities
    private BorderPane root = new BorderPane();
    private Pane gamePane = new Pane();
    private Label scoreLabel = new Label("Score: 0");

    private int score = 0; // Used for score-keeping
    private int enemySpeed = 4; // Default speed of enemies
    private int previousTick; // Used for delay
    private boolean isPaused = false; // Used for pausing the game

    // Used for player movement and key-press management
    private boolean _w_key_down = false;
    private boolean _s_key_down = false;
    private boolean _space_key_down = false;
    private boolean _escape_key_down = false;

    // Game time counter declaration (used for the timing and delay of many objects)
    private int t = 0;

    // Creates a bunch of different sprites using the Sprite class
    private Sprite player = new Sprite(150, 300, 45, "player", "/images/Tank1.png");
    private Sprite background = new Sprite(0, 0, 800, "background", "/images/Background.png");
    private Sprite pause = new Sprite(272, 272, 272, "pause", "/images/Pause2.png");
    private Sprite gameover = new Sprite(272, 240, 272, "gameover", "/images/GameOver.png");
    private Sprite spaceToContinue = new Sprite(180, 500, 64, "spaceToContinue", "/images/Continue.png");
    private Sprite explosion = new Sprite(64, 64, 64, "explosion", "images/Explosion.png");

    // Creates an animation loop that updates the various mechanics of the game
    AnimationTimer timer = new AnimationTimer() {
        @Override
        public void handle(long now) {
            update(); // Runs every 16 ms
        }
    };

// METHODS

    // Used to create and update the objects on the javafx Pane
    private Parent createContent() {
        scoreLabel.setFont(Font.font("Verdana",30));
        root.setTop(scoreLabel);
        BorderPane.setAlignment(scoreLabel, Pos.CENTER);
        root.setBackground(new Background(new BackgroundFill(Color.CORNFLOWERBLUE, null, null)));

        // Sets the size of the application window
        gamePane.setPrefSize(800, 800);

        // Used many times throughout code to add sprites visually
        gamePane.getChildren().add(background);
        gamePane.getChildren().add(player);

        // Starts the AnimationTimer and adds the Pane to the BorderPane
        timer.start();
        root.setCenter(gamePane);
        return root;
    }

```

Collisions

```
274         // Removes sprites when dead
275         gamePane.getChildren().removeIf(n -> {
276             Sprite s = (Sprite) n;
277             return s.dead;
278         });
279 
280         // Removes sprites when off-screen
281         gamePane.getChildren().removeIf(n -> {
282             Sprite s = (Sprite) n;
283             return (s.getTranslateX() < -100);
284         });
285         gamePane.getChildren().removeIf(n -> {
286             Sprite s = (Sprite) n;
287             return (s.getTranslateX() > 900);
288         });
289     }
290 
291     // Sets up the javafx stage
292     @Override
293     public void start(Stage stage) throws Exception {
294         Scene scene = new Scene(createContent());
295         Image Tank1 = new Image("/images/Tank1.png");
296         Image Tank2 = new Image("/images/Tank2.png");
297 
```

```
212     // Does many of the core game mechanics mostly through the interaction of sprites with other sprites
213     private void update() {
214         addEnemies(); // Puts enemies on screen
215         movePlayer(); // Does things with the player when keys are pressed on the keyboard
216         gameOver(); // Does stuff when player dies
217 
218         // Game time incrementor
219         t += 1;
220 
221         // Increases enemy speed over time
222         if (t % 1000 == 0 && !player.dead) {
223             enemySpeed += 1;
224         }
225 
226         // Iterates through all sprites and does collision and mechanics for each type
227         sprites().forEach(s -> {
228             switch (s.type) {
229 
230                 // Player bullet collision
231                 case "playerbullet":
232                     s.moveRight(8);
233                     // Player bullet to Duck collision
234                     sprites().stream().filter(e -> e.type.equals("enemy")).forEach(enemy -> {
235                         if (s.getBoundsInParent().intersects(enemy.getBoundsInParent())) {
236                             enemy.dead = true;
237                             s.dead = true;
238                             score += 1;
239                             scoreLabel.setText("Score: " + score);
240                         }
241                     });
242                     // Player bullet to Skyscraper collision
243                     sprites().stream().filter(e -> e.type.equals("obstacle")).forEach(obstacle -> {
244                         if (s.getBoundsInParent().intersects(obstacle.getBoundsInParent())) {
245                             s.dead = true;
246                         }
247                     });
248                     break;
249 
250                 // Enemy collision and shooting mechanic
251                 case "enemy":
252 
253                     s.moveLeft(enemySpeed);
254                     if (s.getBoundsInParent().intersects(player.getBoundsInParent()) && !player.dead) {
255                         player.dead = true;
256                         previousTick = t;
257                     }
258                     if (Math.random() < 0.1) {
259                         enemyAnimation(s);
260                     }
261                     break;
262 
263                 // Obstacle (skyscraper) collision
264                 case "obstacle":
265                     s.moveLeft(enemySpeed - 1);
266                     if (s.getBoundsInParent().intersects(player.getBoundsInParent()) && !player.dead) {
267                         player.dead = true;
268                         previousTick = t;
269                     }
270                     break;
271             }
272         });
273 
274         // Removes sprites when dead
275         gamePane.getChildren().removeIf(n -> {
276             Sprite s = (Sprite) n;
277             return s.dead;
278         });
```

Enemies & Obstacles

```java
// Creates enemies and puts them on the screen
private void addEnemies() {
    // Duck enemy:
    if (t % 20 == 0) {
        Sprite s = new Sprite(
            (int)gamePane.getWidth() + 100,
            ThreadLocalRandom.current().nextInt(0, ((int)gamePane.getHeight() - 30)),
            30,
            "enemy",
            "/images/Duck1.png"
        );
        // Puts the sprite on the screen
        gamePane.getChildren().add(s);
    }

    // Skyscraper enemy:
    if (t % 200 == 0) {
        Sprite s = new Sprite(
            (int)gamePane.getWidth() + 100,
            ThreadLocalRandom.current().nextInt(0, ((int)gamePane.getHeight() - 150)),
            150,
            "obstacle",
            "/images/Building1.png"
        );
        // Puts the sprite on the screen
        gamePane.getChildren().add(s);
    }
}

// Switches the duck enemy images occasionally
private void enemyAnimation(Sprite s) {
    Image Duck1 = new Image("/images/Duck1.png");
    Image Duck2 = new Image("/images/Duck2.png");

    if (t % 120 <= 25 && s.getImage() != Duck2) {
        s.setImage(Duck2);
    }
    if (t % 120 >= 60 && s.getImage() != Duck1) {
        s.setImage(Duck1);
    }
}

// Used to give Sprite(s) the ability to shoot
private void shoot(Sprite who) {
    Sprite s = new Sprite(
        (int)who.getTranslateX() + (int)who.getBoundsInParent().getWidth(),
        (int)who.getTranslateY(),
        15,
        who.type + "bullet",
        "/images/Projectile.png"
    );
    gamePane.getChildren().add(s);
}
```

```java
// Does many of the core game mechanics mostly through the interaction of sprites with other sprites
private void update() {
    addEnemies(); // Puts enemies on screen
    movePlayer(); // Does things with the player when keys are pressed on the keyboard
    gameOver(); // Does stuff when player dies

    // Game time incrementor
    t += 1;

    // Increases enemy speed over time
    if (t % 1000 == 0 && !player.dead) {
        enemySpeed += 1;
    }

    // Iterates through all sprites and does collision and mechanics for each type
    sprites().forEach(s -> {
        switch (s.type) {

            // Player bullet collision
            case "playerbullet":
                s.moveRight(8);
                // Player bullet to Duck collision
                sprites().stream().filter(e -> e.type.equals("enemy")).forEach(enemy -> {
                    if (s.getBoundsInParent().intersects(enemy.getBoundsInParent())) {
                        enemy.dead = true;
                        s.dead = true;
                        score += 1;
                        scoreLabel.setText("Score: " + score);
                    }
                });
                // Player bullet to Skyscraper collision
                sprites().stream().filter(e -> e.type.equals("obstacle")).forEach(obstacle -> {
                    if (s.getBoundsInParent().intersects(obstacle.getBoundsInParent())) {
                        s.dead = true;
                    }
                });
                break;

            // Enemy collision and shooting mechanic
            case "enemy":

                s.moveLeft(enemySpeed);
                if (s.getBoundsInParent().intersects(player.getBoundsInParent()) && !player.dead) {
                    player.dead = true;
                    previousTick = t;
                }
                if (Math.random() < 0.1) {
                    enemyAnimation(s);
                }
                break;

            // Obstacle (skyscraper) collision
            case "obstacle":
                s.moveLeft(enemySpeed - 1);
                if (s.getBoundsInParent().intersects(player.getBoundsInParent()) && !player.dead) {
                    player.dead = true;
                    previousTick = t;
                }
                break;
        }
    });

    // Removes sprites when dead
    gamePane.getChildren().removeIf(n -> {
        Sprite s = (Sprite) n;
        return s.dead;
    });
```

Score

```
import javafx.animation.AnimationTimer;

public class FlyingTankSimulator extends Application {

// GLOBAL VARIABLES

    // Creates javafx entities
    private BorderPane root = new BorderPane();
    private Pane gamePane = new Pane();
    private Label scoreLabel = new Label("Score: 0");

    private int score = 0; // Used for score-keeping
    private int enemySpeed = 4; // Default speed of enemies
    private int previousTick; // Used for delay
    private boolean isPaused = false; // Used for pausing the game

    // Used for player movement and key-press management
    private boolean _w_key_down = false;
    private boolean _s_key_down = false;
    private boolean _space_key_down = false;
    private boolean _escape_key_down = false;

    // Game time counter declaration (used for the timing and delay of many objects)
    private int t = 0;

    // Creates a bunch of different sprites using the Sprite class
    private Sprite player = new Sprite(150, 300, 45, "player", "/images/Tank1.png");
    private Sprite background = new Sprite(0, 0, 800, "background", "/images/Background.png");
    private Sprite pause = new Sprite(272, 272, 272, "pause", "/images/Pause2.png");
    private Sprite gameover = new Sprite(272, 240, 272, "gameover", "/images/GameOver.png");
    private Sprite spaceToContinue = new Sprite(180, 500, 64, "spaceToContinue", "/images/Continue.png");
    private Sprite explosion = new Sprite(64, 64, 64, "explosion", "images/Explosion.png");

    // Creates an animation loop that updates the various mechanics of the game
    AnimationTimer timer = new AnimationTimer() {
        @Override
        public void handle(long now) {
            update(); // Runs every 16 ms
        }
    };
```

```
    // Does many of the core game mechanics mostly through the interaction of sprites with other sprites
    private void update() {
        addEnemies(); // Puts enemies on screen
        movePlayer(); // Does things with the player when keys are pressed on the keyboard
        gameOver(); // Does stuff when player dies

        // Game time incrementor
        t += 1;

        // Increases enemy speed over time
        if (t % 1000 == 0 && !player.dead) {
            enemySpeed += 1;
        }

        // Iterates through all sprites and does collision and mechanics for each type
        sprites().forEach(s -> {
            switch (s.type) {

                // Player bullet collision
                case "playerbullet":
                    s.moveRight(8);
                    // Player bullet to Duck collision
                    sprites().stream().filter(e -> e.type.equals("enemy")).forEach(enemy -> {
                        if (s.getBoundsInParent().intersects(enemy.getBoundsInParent())) {
                            enemy.dead = true;
                            s.dead = true;
                            score += 1;
                            scoreLabel.setText("Score: " + score);
                        }
                    });
                    // Player bullet to Skyscraper collision
                    sprites().stream().filter(e -> e.type.equals("obstacle")).forEach(obstacle -> {
                        if (s.getBoundsInParent().intersects(obstacle.getBoundsInParent())) {
                            s.dead = true;
                        }
                    });
                    break;

                // Enemy collision and shooting mechanic
                case "enemy":

                    s.moveLeft(enemySpeed);
                    if (s.getBoundsInParent().intersects(player.getBoundsInParent()) && !player.dead) {
                        player.dead = true;
                        previousTick = t;
                    }
                    if (Math.random() < 0.1) {
                        enemyAnimation(s);
                    }
                    break;

                // Obstacle (skyscraper) collision
                case "obstacle":
                    s.moveLeft(enemySpeed - 1);
                    if (s.getBoundsInParent().intersects(player.getBoundsInParent()) && !player.dead) {
                        player.dead = true;
                        previousTick = t;
                    }
                    break;
            }
        });

        // Removes sprites when dead
        gamePane.getChildren().removeIf(n -> {
            Sprite s = (Sprite) n;
            return s.dead;
        });
```

Shooting

```
133
134     // Used to give Sprite(s) the ability to shoot
135     private void shoot(Sprite who) {
136         Sprite s = new Sprite(
137             (int)who.getTranslateX() + (int)who.getBoundsInParent().getWidth(),
138             (int)who.getTranslateY(),
139             15,
140             who.type + "bullet",
141             "/images/Projectile.png"
142         );
143         gamePane.getChildren().add(s);
144     }
145
146     // When keys are down, the player moves, the tank shoots, and resets things after the player dies and presses space
147     private void movePlayer() {
148         if (_w_key_down && player.getTranslateY() > 0 && !isPaused) {
149             player.moveUp();
150         }
151         if (_s_key_down && player.getTranslateY() + player.getFitHeight() < gamePane.getHeight() && !isPaused) {
152             player.moveDown();
153         }
154         if (_space_key_down && !player.dead && !isPaused && t >= previousTick + 10) {
155             shoot(player);
156             previousTick = t;
157         }
158         if (_space_key_down && player.dead && t >= previousTick + 80) {
159             reset();
160         }
161     }
162
```

```
212     // Does many of the core game mechanics mostly through the interaction of sprites with other sprites
213     private void update() {
214         addEnemies(); // Puts enemies on screen
215         movePlayer(); // Does things with the player when keys are pressed on the keyboard
216         gameOver(); // Does stuff when player dies
217
218         // Game time incrementor
219         t += 1;
220
221         // Increases enemy speed over time
222         if (t % 1000 == 0 && !player.dead) {
223             enemySpeed += 1;
224         }
225
226         // Iterates through all sprites and does collision and mechanics for each type
227         sprites().forEach(s -> {
228             switch (s.type) {
229
230                 // Player bullet collision
231                 case "playerbullet":
232                     s.moveRight(8);
233                     // Player bullet to Duck collision
234                     sprites().stream().filter(e -> e.type.equals("enemy")).forEach(enemy -> {
235                         if (s.getBoundsInParent().intersects(enemy.getBoundsInParent())) {
236                             enemy.dead = true;
237                             s.dead = true;
238                             score += 1;
239                             scoreLabel.setText("Score: " + score);
240                         }
241                     });
242                     // Player bullet to Skyscraper collision
243                     sprites().stream().filter(e -> e.type.equals("obstacle")).forEach(obstacle -> {
244                         if (s.getBoundsInParent().intersects(obstacle.getBoundsInParent())) {
245                             s.dead = true;
246                         }
247                     });
248                     break;
249
250                 // Enemy collision and shooting mechanic
251                 case "enemy":
252
253                     s.moveLeft(enemySpeed);
254                     if (s.getBoundsInParent().intersects(player.getBoundsInParent()) && !player.dead) {
255                         player.dead = true;
256                         previousTick = t;
257                     }
258                     if (Math.random() < 0.1) {
259                         enemyAnimation(s);
260                     }
261                     break;
262
263                 // Obstacle (skyscraper) collision
264                 case "obstacle":
265                     s.moveLeft(enemySpeed - 1);
266                     if (s.getBoundsInParent().intersects(player.getBoundsInParent()) && !player.dead) {
267                         player.dead = true;
268                         previousTick = t;
269                     }
270                     break;
271             }
272         });
273
274         // Removes sprites when dead
275         gamePane.getChildren().removeIf(n -> {
276             Sprite s = (Sprite) n;
277             return s.dead;
278         });
```

User Input (Movement, Shooting, and Pause)

```
298        // Detects when a key is pressed down (or held) on the keyboard
299        scene.setOnKeyPressed(p -> {
300            switch (p.getCode()) {
301                case W:
302                    _w_key_down = true;
303                    break;
304                case S:
305                    _s_key_down = true;
306                    if (player.getImage() != Tank2) {
307                        player.setImage(Tank2);
308                    }
309                    break;
310                case SPACE:
311                    _space_key_down = true;
312                    break;
313                case ESCAPE:
314                    // Pauses/stops the AnimationTimer when ESCAPE is pressed
315                    if (!isPaused && !player.dead) {
316                        timer.stop();
317                        isPaused = true;
318                        gamePane.getChildren().add(pause);
319                    }
320                    break;
321                default:
322                    break;
323            }
324        });
325
326        // Detects when a key is released on the keyboard
327        scene.setOnKeyReleased(r -> {
328            switch (r.getCode()) {
329                case W:
330                    _w_key_down = false;
331                    break;
332                case S:
333                    _s_key_down = false;
334                    if (player.getImage() != Tank1) {
335                        player.setImage(Tank1);
336                    }
337                    break;
338                case SPACE:
339                    _space_key_down = false;
340                    break;
341                case ESCAPE:
342                    // Resumes/starts the AnimationTimer when ESCAPE is released a second time
343                    if (isPaused && _escape_key_down) {
344                        timer.start();
345                        isPaused = false;
346                        gamePane.getChildren().remove(pause);
347                        _escape_key_down = false;
348                    }
349                    if (isPaused) {
350                        _escape_key_down = true;
351                    }
352                    break;
353                default:
354                    break;
355            }
356        });
```

```
364
365    // SPRITE CLASS
366
367        // Reusable Sprite class for adding visual objects to the game
368        public static class Sprite extends ImageView {
369            boolean dead = false;
370    //      double x_velocity = 0;
371    //      double y_velocity = 0;
372            final String type;
373            final String location;
374
375            // Initializing the Sprite class
376            Sprite(int x, int y, int size, String type, String location) {
377                super(location);
378                this.location = location;
379                this.type = type;
380                setTranslateX(x);
381                setTranslateY(y);
382                this.setFitHeight(size);
383                this.setPreserveRatio(true);
384            }
385
386            // Sprite movement
387            void moveLeft(int amount) {
388                setTranslateX(getTranslateX() - amount);
389            }
390            void moveRight(int amount) {
391                setTranslateX(getTranslateX() + amount);
392            }
393            void moveUp() {
394                setTranslateY(getTranslateY() - 8);
395            }
396            void moveDown() {
397                setTranslateY(getTranslateY() + 8);
398            }
399        }
400
```

```
146    // When keys are down, the player moves, the tank shoots, and resets things after the player dies and presses space
147    private void movePlayer() {
148        if (_w_key_down && player.getTranslateY() > 0 && !isPaused) {
149            player.moveUp();
150        }
151        if (_s_key_down && player.getTranslateY() + player.getFitHeight() < gamePane.getHeight() && !isPaused) {
152            player.moveDown();
153        }
154        if (_space_key_down && !player.dead && !isPaused && t >= previousTick + 10) {
155            shoot(player);
156            previousTick = t;
157        }
158        if (_space_key_down && player.dead && t >= previousTick + 80) {
159            reset();
160        }
161    }
162
```

Game Over

```
163     // Resets values and clears the screen of objects
164     private void reset() {
165         score = 0;
166         t = 0;
167         enemySpeed = 4;
168         previousTick = 0;
169         player.setTranslateY(300);
170         scoreLabel.setText("Score: " + score);
171         gamePane.getChildren().clear();
172         gamePane.getChildren().add(background);
173         player.dead = false;
174         gamePane.getChildren().add(player);
175     }
176
177     // Helper function for update() and gameOver() that is used to parse through all the sprites
178     private List<Sprite> sprites() {
179         return gamePane.getChildren().stream().map(n -> (Sprite)n).collect(Collectors.toList());
180     }
181
182     // Does stuff when player dies
183     private void gameOver() {
184         // Game Over Screen/Press Space to Continue and explosion mechanic
185         if (player.dead) {
186             if (t >= previousTick + 40 && !sprites().contains(gameover)) {
187                 gamePane.getChildren().add(gameover);
188             }
189             if (t >= previousTick + 150 && !sprites().contains(spaceToContinue)) {
190                 gamePane.getChildren().add(spaceToContinue);
191             }
192             if (!sprites().contains(explosion)) {
193                 explosion.setTranslateX(player.getTranslateX());
194                 explosion.setTranslateY(player.getTranslateY());
195                 gamePane.getChildren().add(explosion);
196             }
197             // Replaces player Sprite with Dead Explosion Sprite
198             if (t >= previousTick + 60 && sprites().contains(explosion)) {
199                 gamePane.getChildren().remove(explosion);
200             }
201         }
202
203         // Puts the Game Over and Press Space to Continue text in front of everything
204         if (player.dead && sprites().contains(gameover) && sprites().contains(spaceToContinue)) {
205             gamePane.getChildren().remove(gameover);
206             gamePane.getChildren().remove(spaceToContinue);
207             gamePane.getChildren().add(gameover);
208             gamePane.getChildren().add(spaceToContinue);
209         }
210     }
```

Time for Reflection

Individual Responsibilities

Krissy - Designer: recorded project demos and created almost all sprites (tank, birds, skyscrapers, background, explosion, pause text, game over text, and ‘press space to continue’ text)

Zach - Organizer: led with management of the group and worked on group submissions

Josh - Main Coder: did the majority of the coding and commenting, solving errors and improving on some existing code from others

Chelsea - Researcher: researched versions of the game and implemented the game over mechanic and the pause sprite

New Features Since Midterm Presentation

- Sprites
  - The blue square is now a tank, skyscrapers are in place of the blocks, and we added duck enemies that can be shot
- Smooth movement
  - Multiple keys can be pressed and held at once with no troubles and the player looks less jerky when moving up and down
- Scoring
  - The player accumulates points when killing ducks
- Pause screen
  - When the user presses ‘escape’, the game will stop and bring up the ‘pause’ text, and then will resume when pressed again
- Game over screen
  - When the player dies, many images are put on screen, including the text ‘game over’

Changes From the Initial Design

- The original code was that of the game “Flappy Bird”
- The goal changed to make the game more distinct
- The game was originally going to be just a tank that dodged obstacles, but instead morphed into a tank that shoots ducks and dodges floating skyscrapers, which speed up over time

Major Challenges and Solutions

- Trying to do things with no experience
- Implementing various mechanics
  - Pause menu
  - Delay between various things (such as shooting bullets)
  - Smooth player movement
  - Score counter

Most Fun Part of the Project

- Implementing comical elements into the game
- Playing the game
- Looking back at what we've learned through it and having the experience to apply toward future endeavors

The End...?

Improvements have been made after the semester ended!
See if you can spot all the changes!

https://youtu.be/sH3YWWQ56bo