Victor Feng He
CISC 3667 - Dr. Devorah Kletenik

1. layered background (background /foreground), at least one image, and a player-controlled sprite
2. an balloon sprite with automatic movement
3. the ability for the player to shoot pins at the enemy
4. collision detection of pins, using tags so that a player does not pop himself with his own bullets
5. sound effect on collisions
6. displayed score for player
7. increasing size of balloon and impact on score
8. at least one distractor
9. at least three levels in increasing order of difficulty. Document the difficulty of each level in the directions.
10. scene transitions: Every time that your player pops the balloon, the game should transition to the next level. Every time the balloon gets too big and disappears, the current level should be restarted. OR a slightly different version that you chose to implement.
11. Clear directions (include the basics of each level
12. settings, including a volume setting with a slider
13. menu
14. pause/resume and link back to menu
15. some other UI (dropdown, toggle, input)
16. a data item that persist from scene to scene
17. a second data item that persists
18. high scores (at least 5, presented in order)
19. animation #1
20. animation #2
21. Extra credit: difficulty selection by player (with documentation about difficulty)

Game Description
https://github.com/DeanEvil/AlienPopper

Throughout development, I had initially wanted to include multiple balloons in the final level to act as a final challenge. However, when trying to implement scripts that would start the level transition once all balloons in a level were gone, the transition would happen even if only one balloon out of two was destroyed. Thus, I opted to make a sort of obstacle course for the final level instead (I am most proud about this part, since the level felt like it was difficult enough for the player to retry a few times, but not so hard that it would be frustrating). I've also added background music to the game to make it seem a bit more lively and a volume slider in the settings to better adjust the BGM volume. I believe that the uniqueness of this game comes from how the player is restricted by physics and the projectile is also affected by gravity. This can make for an interesting interaction with how the player needs to quickly but also navigate through levels to successfully get a high score.

Checklist of Components

1. **Yes** - The background is the picture with three green pipes, a sky, and a dirt ground (all pixelated). The foreground is made up of invisible colliders that are placed at the top and right of the screen, as well as smaller colliders that cover the top of each pipe to act as platforms for the soldier sprite (player) to stand on.
2. **Yes** - Each of the three levels has a purple balloon sprite that moves up and down; can usually be found to the right of the player character.
3. **Yes** - Player character has the ability to shoot bullets (pins) at the purple balloons. They spawn in front of the player character.
4. **Yes** - Each bullet has a 2d box collider and a rigid body that is affected by gravity. Bullets can be found by pressing the "E" key so that bullets can be spawned by an invisible GameObject called "bulletSpawn." The invisible object makes it so that bullets don't randomly spawn behind the player or hit the player.
5. **No** - There is code to play the audio for a pop sfx in the PopBalloon() method of the BalloonMovement script, but the following code makes it so that scene transition takes place before sound can be played.

```
foreach (Collider2D collider in colliders)
{
    if (collider.CompareTag("Bullet"))
    {
        int level = SceneManager.GetActiveScene().buildIndex;
        PopBalloon();
        SceneManager.LoadScene(level + 1);
    }
}
```

6. **Yes** - There is a text box for Score in green font and can be found at the left side of the screen in each level.
7. **Yes** - This component is present; if enough time passess, the balloon will grow noticeably bigger and its 2d box collider component will also grow accordingly.

8. **Yes -** The distractors are black-colored walls, which can be found in the second and third levels acting as obstacles for the player to navigate around.
9. **Yes -** The three levels of difficulty are present; Documentation of each level's difficulty and how to navigate each level can be found by clicking on the Info button in the MainMenu screen of the game.
10. **Yes -** Game transitions to the next level when a balloon is popped. The balloon also disappears when its size reaches 4 and the level is restarted. Player is also able to manually restart the level by pressing the "R" key.
11. **Yes -** Directions are included in the Info screen of the game.
12. **Yes -** This can be found by clicking on the settings button of the MainMenu screen. Volume slider is also functional, controlling the volume of background music.
13. **Yes -** This can be found when you first launch the game or when you click on the menu button in any of the levels as well as in the Scoreboard screen.
14. **Yes -** This is present for all levels of the game.
15. **Yes -** There is an input field for the player to enter their name in the MainMenu screen. There's also a grid layout UI for the Scoreboard screen
16. **Yes -** The name input by the player is a data item that persists through the whole game as "Hello [blank]!" (with [blank] being the username).
17. **Yes -** Another data item is the total score of the player, which updates after every balloon that is popped. (Max score is 15).
18. **Yes -** The Scoreboard screen which can be accessed after the final level of the game can display up to 10 high scores in descending order (in full screen).
19. **Yes -** The soldier character has a walking animation when the player holds down the "A" or "D" keys.
20. **Yes -** The soldier character has a shooting animation when the player presses the "E" key.
21. **No**

Citations of Resources Used for Programming/Unity

- Used this video to make my Scoreboard screen:
  ▶ Creating Scoreboard in Unity | Under 5 minutes!
  Also used and modified Dr. Kletenik's ScoreKeeper and Persistent Data scripts found in the Unity-Fall-2023 project on Github in order to connect player score and name to my Scoreboard.
- Used the PauseResume script and ButtonFunctions script from Dr.Kletenik's project to implement the Pause/Resume and Menu components of my game.
- Referred to Dr.Kletenik's Unity-Fall-2023 project that I pulled from her Github repository for the overall design of my game.
- Watched this video to help with controlling the volume of my music in SettingsScene of my game: ▶ Unity Audio: How To Make a UI VOLUME SLIDER (Saving & Loading)
- Used ChatGPT to help write my script for BalloonMovement:

You

I have a Balloon Sprite in Unity. Write a new script that controls its movement. This will not be controlled by user input, so make it fully automatic -- it should move up or down across the screen. Note that you will want to check if it collides with an invisble 2d box collider (use the transform.position attribute); if it does, flip it around and send it back in the opposite direction. In addition, make sure that the balloon is increasing in size as time goes by. If the balloon size increases past a certain size, the level restarts.

ChatGPT

Certainly! Below is a simple script in C# that you can attach to your Balloon Sprite in Unity. This script makes the Balloon move up and down, check for collisions with an invisible 2D box collider, flip its direction upon collision, and increase in size over time. If the Balloon size exceeds a certain limit, it triggers a level restart.

```csharp
using UnityEngine;

public class BalloonController : MonoBehaviour
{
    public float moveSpeed = 2.0f;         // Speed of the Balloon's vertical movement
    public float maxSize = 5.0f;           // Maximum size of the Balloon before level
    public float sizeIncreaseRate = 0.1f;  // Rate at which the Balloon size increases

    private int moveDirection = 1;         // 1 for moving up, -1 for moving down

    void Update()
    {
        MoveBalloon();
        CheckCollision();
        IncreaseSize();
        CheckSizeLimit();
    }

    void MoveBalloon()
    {
        // Move the Balloon vertically
        transform.Translate(Vector3.up * moveDirection * moveSpeed * Time.deltaTime);
    }
}
```

```
CheckCollision()

// Check if the Balloon collides with an invisible box collider
// Adjust the size of the box collider based on your game design
// For example, you can use an empty GameObject with a Box Collider 2D component as the

// Replace "InvisibleCollider" with the actual name of your invisible collider GameObje
GameObject invisibleCollider = GameObject.Find("InvisibleCollider");

if (invisibleCollider != null)
{
    // Get the position of the Balloon and the size of the invisible collider
    Vector3 balloonPosition = transform.position;
    float colliderSizeY = invisibleCollider.GetComponent<BoxCollider2D>().size.y / 2;

    // Check if the Balloon collides with the top or bottom of the invisible collider
    if (balloonPosition.y + colliderSizeY >= invisibleCollider.transform.position.y ||
        balloonPosition.y - colliderSizeY <= invisibleCollider.transform.position.y)
    {
        // Reverse the direction of the Balloon when collision occurs
        moveDirection *= -1;
    }
}


IncreaseSize()

// Increase the size of the Balloon over time
transform.localScale += new Vector3(sizeIncreaseRate, sizeIncreaseRate, 0) * Time.delta

CheckSizeLimit()
```

- Used ChatGPT to help with activating the shooting animation as the player shoots:

V **You**
I have a Movement script attached to my player avatar. The avatar has an idle, shooting, and walking animation. Considering this, How do I get the avatar to shoot pins and trigger the shooting animation when the "e" button is pressed? I also want the avatar to move left and right with the walking animation. When not doing either of these things, the default state should be the idle animation.

PlayerController Script:

```csharp
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    public float moveSpeed = 5f;
    public GameObject pinPrefab;
    public Transform pinSpawnPoint;
    public float pinForce = 10f;

    private Animator animator;

    void Start()
    {
        animator = GetComponent<Animator>();
    }

    void Update()
    {
        HandleMovementInput();
        HandleShootingInput();
    }

    void HandleMovementInput()
    {
        float horizontalInput = Input.GetAxis("Horizontal");

        // Set the horizontal input value to control the walking animation
        animator.SetFloat("Speed", Mathf.Abs(horizontalInput));
```

```csharp
        transform.Translate(movement * moveSpeed * Time.deltaTime);


        // Flip the player sprite when moving left
        if (horizontalInput < 0)
        {
            transform.localScale = new Vector3(-1, 1, 1);
        }
        // Flip the player sprite when moving right
        else if (horizontalInput > 0)
        {
            transform.localScale = new Vector3(1, 1, 1);
        }
    }


    void HandleShootingInput()
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            Shoot();
        }
    }


    void Shoot()
    {
        // Trigger the shooting animation
        animator.SetTrigger("Shoot");

        // Instantiate a new pin at the pinSpawnPoint
        GameObject pin = Instantiate(pinPrefab, pinSpawnPoint.position, pinSpawnPoint
```

↓

```
            // Apply force to the pin in the forward direction
            Rigidbody2D pinRb = pin.GetComponent<Rigidbody2D>();
            if (pinRb != null)
            {
                pinRb.AddForce(pinSpawnPoint.right * pinForce, ForceMode2D.Impulse);
            }
        }
    }
```

Animator Setup:

1. Create an Animator Controller asset in your project.
2. Open the Animator window and create animation states for "Idle," "Walking," and "Shooting."
3. Set up transitions between these states. Create transitions from "Idle" to "Walking" based on the "Speed" parameter, and from "Idle" to "Shooting" and "Walking" to "Shooting" based on the "Shoot" trigger.
4. Set up parameters in the Animator:
   - Create a trigger parameter named "Shoot."
   - Create a float parameter named "Speed."
5. Assign the animations to the corresponding states.

This setup assumes that you have idle, walking, and shooting animations, and the "Speed" parameter controls the transition between idle and walking animations. The "Shoot" trigger controls the transition to the shooting animation.

Make sure to adjust animation speeds, transitions, and other parameters based on your specific animations and game requirements.