PROGRAMMING REQUIREMENTS During Proof of Concept, the programming team is trying to prove they have the technical capability to develop all requirements for the game and most importantly the hook mechanic. It will be crucial to prototype and get a working version of any technically challenging elements so the team can be assured that the game can be created.

DEVELOPMENT PLAN Create a document that outlines a plan for developing your hook mechanic and other required systems for your game, the document should outline:

HOOK: Dash
Mechanics:

- **Movement**
- **Jumping**
- **Gate**
- **Collectables**
- **Ink Arms**
- **Rising Ink**
- **Pop up**

• What new components/systems you think will need to be added to the main player and to the environment for the mechanic to function properly. Outline what variables the component will need. (Ex. for a shooting mechanic, aim speed, projectile, ammo, etc.)

DASH Requirements

Player

1. **Speed of Dash [float]**
2. **Direction [float] ~ based on input values (0.0 to 0.1)**
3. **Length(time) of dash [float]**
4. **Dashing [boolean]**

MOVE Requirements

Player

1. **Player move speed [float]**
2. **Colliding with ground [boolean] ~ to decide if gravity should be applied**
3. **Current Friction [float] ~ the friction they are facing from the ground**
4. **Acceleration time [float] ~ time in seconds to get up to speed**

JUMP Requirements

Player

1. **Jump Speed [float]**

2. **Jump cancel [float] ~ reduce player speed when they release the jump button**
3. **Forget Jump [float] ~ the amount of time in which the game queues the player's jump command**

GATE Requirements

Gate

1. **Speed the gate will open at [float]**

COLLECTABLES Requirements

Player

1. **Flower count [float]**
2. **HasKey [boolean]**

RISING INK Requirements

Rising Ink

1. **Speed it will rise at [float]**
2. **Move [boolean] - when set to true the ink will rise**
3. **Wait [float] - the amount of time the timer function will wait**

UI Dialogue Requirements

1. **…**

Sound Triggers Requirements

1. **OnTriggerEnter [Boolean]**

Shaders Requirements

1. **This is covered in a different document**

Pop Up Requirements

1. **Pop [boolean]**
2. **Pop_Speed [float]**

Ink Arms

1. **ANCHOR [Transform] ~ stores the anchor of the arm**
2. **Arm_speed [float] ~ speed at which the arm moves**
3. **Arm_reach [float] ~ the distance the hand is allowed to move from the anchor**
4. **Return [bool] ~ if the player is out of range, return into the ink**

• Develop a plan in terms of what order the components will be created and who will be working on what aspects of each one

1. **Movement (Brendan, all aspects)**
2. **Jumping (Brendan, all aspects)**
3. **Dash (Brendan, all aspects)**
4. **Pop-ups (Brendan, all aspects)**
5. **Collectables (Ian, all aspects)**
6. **Gate (Ian, all aspects)**
7. **Ink Arms (Brendan, all aspects)**
8. **Rising Ink (Ian, all aspects)**
9. **UI Dialogue (Aiden, all aspects)**
10. **Shaders (Patrick, all aspects)**
11. **Sound Triggers (Ezra, all aspects)**

• Each programmer should outline their plan for developing the different components, and description of how they think it will work, be as detailed as you can at this stage and possibly include pseudocode or algorithm ideas

Brendan

Movement

When it comes to movement, It will require three steps,

1. **Acceleration**

Acceleration will work something like this

Counter += 1f / (FPS * Time[Seconds])
//This counter represents time passing in the unity update function.

Current_speed = [Basically $\mathbf{2^x}$ where x is the counter above]
//This increases the player's speed at an inverse logarithmic speed

2. **Max Speed**

If (Current_speed.x >= MOVE_SPEED) then
Current_speed = [Basically the max speed]

3. **Deceleration**

friction(){
    If the player is moving
        Reduce their speed
}

Jumping

If (player is on the ground **AND** jump is queued[i.e., has been pressed recently] then
    If (player releases the jump button) then
        Current_speed.y = 0 //effectively
    If (jump queued) then
        Jump queued = false
        Current_speed.y = JUMP_SPEED
[else where every call Current_speed.y will be subtracted by (GRAVITY/FPS)]

Dash

When the player presses the dash key
    Disable their movement using a timer in a separate function

Using Unity's Input.GetAxisRaw, we can determine which direction the player is pointing.

Dash the player in the direction they are facing

If the direction is angled, then the dash is split between the x and y axis to keep the dash balanced.

Pop-Ups

When the player enters a trigger activates pop boolean

If (pop **AND** transform.rotation.x < 0) then
    transform.Rotate(1f/(FPS*POP_SPEED),0,0)

Ink Arms

So this will effectively be a game object at the end of two boxes, these objects are all hinged together. So the script will be attached to the game object at the end. Because a MoveTowards will be applied to the object, causing the InkArms to look arm-like. There will be a method called Stretched() which will determine if the arms are being stretched at a certain distance from their anchor, using Pythagorean theorem. The arm only starts to move once the player has entered the arms trigger.

If the player is outside of the trigger, the arm will retract back into the ink

Ian

Collectables

The collectables in our game consist of the keys to open corresponding gates as well as flowers. Each collectable will show on the player when they pick it up. To have this happen I will have a key or flowers attached to the player but will be hidden when the variable is false or 0. Then it will simply be a script that unhides the object that is attached to the player when they pick up the collectable in the level. The collectable in the level will then be destroyed.

Key

The key collectable will just have the OnTriggerEnter function so that if the player collides with the key, hasKey is set to true and the key gameObject is destroyed i.e.,

```
void OnTriggerEnter(Collider collider)
{
    if (collider.gameObject.tag == "Player")
    {
        PrinceVariables.hasKey = true;
        Destroy(gameObject);
    }
}
```

The key collectable will then hide or unhide on the player script depending on if the hasKey variable is true or false i.e.,

```
void Update()
If (hasKey = true) then
        Gameobject.renderer = false (GameObject would be the Key in this
example)
else
        GameObject.renderer = true
```

Flowers

The flowers code will be pretty much identical to the keys, but each flower will have a specific variable (like hasRed, or hasBlue) so that if the player picks them up in a different order I can just hide or unhide the specific flower they picked up.

Gate

The gate will be a pretty simple script. It will have a OnTriggerEnter function so that when the player collides with the gate it will see if the player has a key. If the player has a key it sets the player's hasKey variable to false and moves the gate up using transform.Translate. This moves the game object, it has 3 parameters it takes in being the speed it will move on each axis, x,y,z. The speed at which the gate moves up will be a float named moveSpeed.

```
void OnTriggerEnter(Collider collider)
{
   if (collider.gameObject.tag == "Player" && PrinceVariables.hasKey == true)
   {

      PrinceVariables.hasKey = false;

      transform.Translate(0, moveSpeed * Time.deltaTime, 0);

   }
}
```

Rising Ink

The rising ink code will be done in a similar way to the gate code above. The ink will have a collider on the actual ink object and when the player enters the collider it will kill them, sending them back to the last checkpoint.

It will have a boolean variable called move set to false automatically. There will be a separate game object in front of the rising ink and when the player collides with it the timer function will be called, making the script wait a couple seconds (set by a variable) so the player can get above the ink and start platforming, then the rising ink script will be referenced and will set move to true.

```
onCollision
if (Id is “player”) then
```

```
Call timer function
risingInk.move = true;
```

In the update function of the rising ink script, if move is set to true have the ink translate upwards just like the gate's code above.

The speed it rises at will be a very slow speed compared to the gate so that the player can out platform it without it being too difficult.

Aidan

UI

The way I plan on doing UI is very simplistic. Our game does not require maps or health bars as it is linear and you only have one life. This means we can keep it minimalistic to keep immersion. However, I plan on using the UI to add text above the characters at points as if they were reading a page of a story book. The way I plan on achieving this is through box colliders I will set as triggers to pop up the corresponding text on the plane for the player to read and understand the story, as well as to sell the idea that they are playing a pop up storybook.

Dialogue Triggers

The Dialogue triggers will work similar to the pop up code (reference the pop up code above before continuing). The dialogue will be short messages that, when the player hits a box collider set up to be a trigger, will appear next to the NPC on a popsicle stick.

Patrick

Shaders

Refer to the art style overview for description of intended shaders and their purpose.

The shader for the ink will use a generated noise map to determine factors such as where on the object ink currently is, the amount of ink on the area(controlled by correlating the noise value and the opacity), and the amount of area that is covered by the ink . The map can be enlarged and moved so that it is covering the intended area of the object. These functions should allow for us to control how the ink spreads across an object, and how it connects neighboring objects via worldspace, so that seamless transitions between objects and the environment are possible.

Ezra

Sound Trigger on Collision

Most sounds will not be playing on loop on launch and therefore need a script to activate them. This simple script is meant to be a catch all for any sounds that need to be played upon activating some kind of trigger. Whether that be gaining a collectable, dying, or jumping. Once the player's makes contact with the collision box of an object that has this script the script will play the activate the audio source attached to the object

```
Public AudioSource playSound;

Void OnTriggerEnter(Collider other)
{
        playSound.Play();
}
```

• Using the maps for each section a created within design and try to identify as many things that will require programming support and document some brief ideas as to how it can be achieved through code and/or built in Unity components. (This is an opportunity to let design know if any ideas are outside the skill range of the team or discuss potential solutions with a professor

Everything shown in the map was already covered in the mechanics created in this document. The mechanics in our game are not very complex so everything is within the skill range of our team