Project Plan – Blue Team

**Introduction:**

Our project will be a synergistic effort that entails combining individually engineered mini-games into a singular program. The games will all be written in Python using various external libraries. Those libraries could include, but are not limited to, PyGame and Arcade. Each member of the team will contribute one entire game module to what we are referring to as our “Mini-Arcade.” Those stand-alone games will then be connected by way of a centralized main screen. The main screen will be a GUI programmed in TKinter to make it interfaceable with Python-constructed games. The structure of the GUI will be a simple to understand button layout, with each button labeled with, and corresponding to, one of the team member’s games.

**Project Organization**:

* **Leader/Organizer**:

For our project, the team has decided as a group to start out with a “round table” type of organizational structure. There will be no stated team leader, with all decisions being made at a team level with a majority rule. We are also looking to a volunteer system, as opposed to one that mandates who does what. While this might necessitate more meetings and communication, the members of Blue Team believe this is the best way of keeping each member interested, motivated, and inclined to do their best work. Should a time come where this structure breaks down and creates more difficulties than benefits, a leader will be voted on and a more rigid “top down” structure instituted. As of the current moment, we do not foresee this to be a necessary step in our development lifecycle.

* **Coder/Developer**:

Each member of Blue Team will be tackling their own coding project insofar as the game they choose to develop and integrate into the overall system. By design, we chose to use a programming language that everyone

felt a level of comfort with specifically for the purpose of allowing the workload to be evenly distributed across the team. The GUI that will connect all the games will be a joint programming venture, with each of the team members coming together after the games are completed and integrating their module.

* **Testing/Quality Assurance**:

Testing and quality assurance will again be a team level activity, with each member’s games being played/tested and the code gone over by the other members of the team. Should a problem be discovered, the original programmer will fix whatever bug is causing the error, and the testing will be redone by the tester who initially caught the error, and another tester after.

* **Analyst**:

If, during the testing phase, a problem is discovered that can’t be reconciled by the original programmer, a second team member will go over the code to attempt to visualize the problem and solution and then walk the original programmer through their potential correction. This, as with the rest of the assigned tasks in Blue Team’s development cycles, will be done on a volunteer basis, most likely decided by which member has the most available time to devote to the problem during that cycle.

* **Designer**:

The term designer is another shared designation in our team. With each mini-game being developed individually, the design of those games will fall to the team member in charge of coding the game. With regards to the overall look of the GUI, there will be a consensus among members of Blue Team as to the end result that is presented to the user.

Below is a chart that shows the organizational structure:

|  |  |  |
| --- | --- | --- |
| **Role:** | **Team Member:** | **Description:** |
| Leader/  Organizer | “Round Table” structure | We have decided as a team to not utilize a team leader, per se, and let everything be done by vote or on a volunteer basis. |
| Coder/  Developer | Alex Johnston, Brandon Kesner,  Makayla Harrison, Craig Hutson | Since everyone will be contributing their own game modules, everyone will take on the role of developer in all phases of our production lifecycle. |
| Tester/  Quality Assurance | Alex Johnston, Brandon Kesner,  Makayla Harrison, Craig Hutson | The games will be tested as they are completed by whichever team member has the most available time during the specific lifecycle. It will be determined on a volunteer basis. |
| Analyst | Volunteer\* | The analyst position will only become necessary if a coding problem evolves that the initial coder can’t reconcile on their own. It will be solved in the same manner that games will be tested. |
| Designer | Blue Team | All members will contribute to the overall look and functionality of the program. |

Risk Analysis: Python Mini-Arcade

This document outlines potential risks and risk mitigation methods for our student software project in SDEV 265. The intention is a simulated real-world development cycle, not simply a class assignment. That requires a stringent level of honesty pertaining to potential obstacles and a detailed level of focus on how to avoid getting derailed and/or how to get things back on track.

## Cost Risk

**What Could Go Wrong:**

We’re not spending money, but this eight-week term gives us a hard deadline to work with, making time our primary currency. If we underestimate how long a portion of the development cycle takes, like debugging movement or sourcing assets, or poorly allocate our resources, we could burn hours we can’t afford to waste and can’t replace.

**How We’ll Work Around It:**

We’ll keep the scope tight and modular. If a feature starts eating up time, we’ll flag it early and either simplify or postpone it. We’ll also use free tools that don’t require setup time such as GitHub, Google Docs, and open asset libraries.

## Schedule Risk

**What Could Go Wrong:**

Life happens. Assignments pile up. Work and family come with their own time requirements. Someone could become ill and miss important milestones. If we fall behind early, it’ll be hard to catch up later, particularly if it is due to an illness that renders the team down a person for a time.

**How We’ll Work Around It:**

We’re setting internal deadlines ahead of the official ones. That gives us breathing room. We’ll also use a shared task board, so everyone knows what’s done and what’s next. If someone’s stuck, we pivot fast and redistribute.

Option one: We adjust the scope, delivering three games instead of four and modifying the game launcher accordingly.

Option two: One or more of the other team members who have their games in good shape and up to date with time constraints can step in offering any and all needed assistance. Shared files and notes can facilitate a smooth hand-off, and a priority can be placed on solid working builds over polished finish products in accordance with Agile development.

## Performance Risk

**What Could Go Wrong:**

The games could lag, break, or just feel clunky. There could be problems unifying the games into one system. If we don’t test enough, we might not catch bugs until it’s too late.

**How We’ll Work Around It:**

We’ll test early and often. Even basic movement functionality will be reviewed before adding more layers. We’ll keep the code clean and modular so fixing one part doesn’t break the rest. And we’ll use simple assets to keep performance smooth.

## Operational Risk

**What Could Go Wrong:**

If roles aren’t clear or communication breaks down, we’ll waste time or duplicate work. Worst case: someone feels left out or overwhelmed.

**How We’ll Work Around It:**

We’ve already defined roles, and we’ll keep checking in to make sure everyone’s comfortable. If someone wants to switch tasks or needs help, we adjust. We’re using GitHub for version control and Google Docs for planning, so structure remains constant.

## Technology Risk

**What Could Go Wrong:**

Libraries like PyGame and Arcade might not work the way we expect. We could hit compatibility issues or spend too long learning them.

**How We’ll Work Around It:**

We’ll start with plain Python. It’s something we’re all fluent in. If we use a library, it’ll be because it saves time, not adds complexity. We’ll test it in a sandbox before committing. If it slows us down, we drop it and go manual.

## Communications Risk

**What Could Go Wrong:**

If we don’t share updates or ask questions, we’ll drift out of sync. Misunderstandings can lead to wasted effort or missed deadlines.

**How We’ll Work Around It:**

We’ll use a shared doc before and after each regularly scheduled meeting to track what’s discussed and what’s next. We’ll keep messages short and clear. If something’s confusing, we ask. No one’s expected to read minds.

## Scope Creep Risk

**What Could Go Wrong:**

We get excited and start adding features, new weapons, levels, animations. Suddenly, the project’s twice as big and half as finished.

**How We’ll Work Around It:**

We’ll lock the core features early. Anything extra goes on a “maybe later” list. If we finish early, great, we can add polish. But we won’t sacrifice stability for flash.

## Skills Resource Risk

**What Could Go Wrong:**

Not everyone’s confident with code, design, or debugging. If someone’s stuck or overwhelmed, it slows the whole team.

**How We’ll Work Around It:**

We match tasks to comfort levels. If someone’s new to coding, they can handle assets or documentation. If someone wants to learn, we pair up and walk through it. No one’s expected to know everything, we build together.

## Final Thoughts

This isn’t just about avoiding problems, it’s about building a system that flexes when things get messy. We’re not aiming for perfection. We’re aiming for progress, clarity, and a game we’re proud to show off. If we stay honest and modular, we’ll get there.

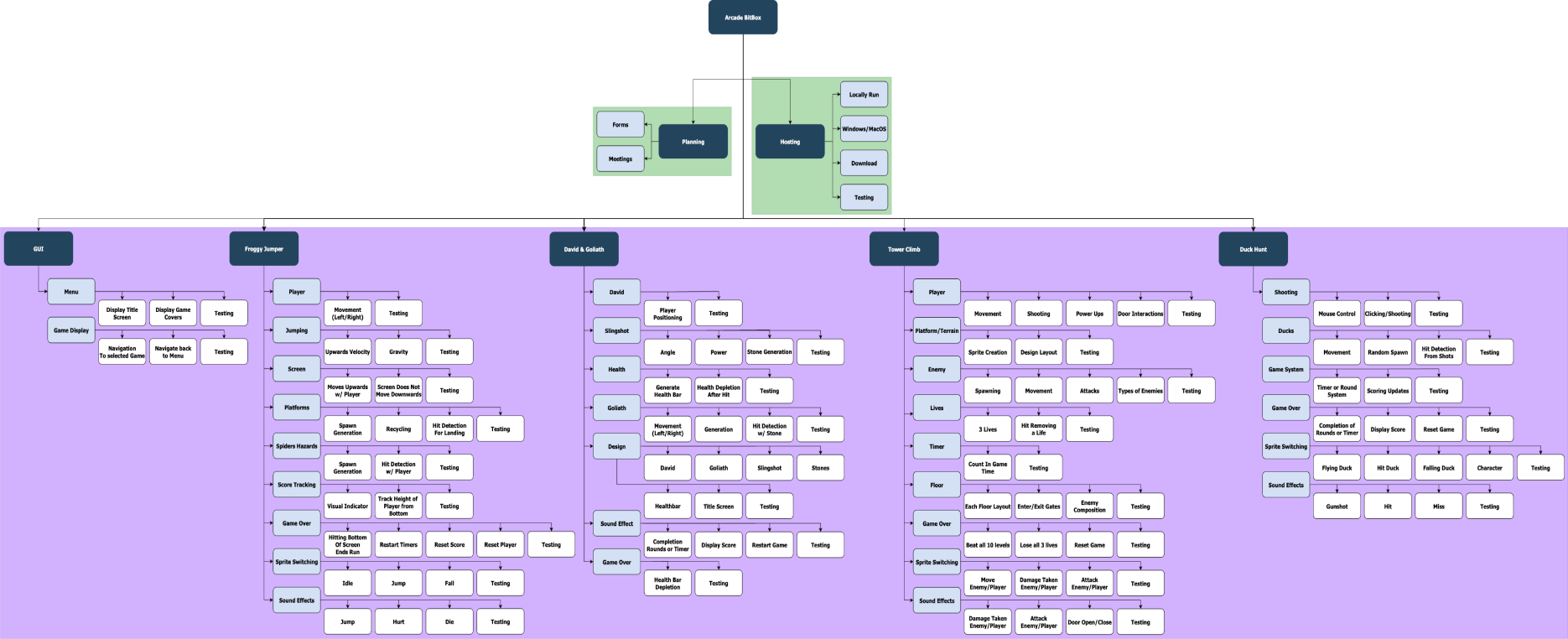
HARD & SOFT REQUIREMENTS:

Hardware Requirements

* CPU: Any modern processor
* RAM: At least 2 GB RAM
* Storage: Less than 100 MB free space
* Graphics: Integrated graphics (Intel HD, AMD Vega etc)

Software Requirements

* Operating System:
  + Windows OS
  + Mac OS
* Python Version:
  + Python 3.8 +
* Libraries Needed:
  + tkinter (built-in with Python on most systems)
* IDE/Editior
  + VS Code
* Programming Language:
  + Python

WORK BREAKDOWN:

Planning

1. Forms
2. Meetings

Hosting

1. Local
2. Windows and MacOS
3. Download
4. Testing

GUI

1. Menu
   1. Display Title Screen
   2. Display Game Covers
   3. Testing
2. Game Display
   1. Navigation to selected
   2. Navigation back to menu
   3. Testing

Tower Climb  
Complete floors to scale the tower

1. Platform/Terrain
   1. Sprite Creation
   2. Design Layout
   3. Testing
2. Player
   1. Movement
   2. Shooting
   3. Pick up power ups
   4. Door Interactions
   5. Testing
3. Enemy
   1. Spawning
   2. Movement
   3. Attacks
   4. Types of Enemies
   5. Testing
4. Lives
   1. 3 lives
   2. Hit Removing a Life
   3. Testing
5. Timer
   1. Count time in game
   2. Testing
6. Floor
   1. Each Floor Layout
   2. Enter/Exit Gates
   3. Enemy Composition
   4. Testing
7. Game Over
   1. Beat floor 10
   2. Lose all 3 lives
   3. Restart Game
   4. Testing
8. Sprite switching
   1. Move - Enemy/Player
   2. Hit - Enemy/Player
   3. Attack - Enemy/Player
   4. Testing
9. Sound Effects
   1. Damage Taken Enemy/Player
   2. Attack Enemy/Player
   3. Door Open/Close
   4. Testing

David vs Goliath

1. David
   1. Player positioned
   2. Testing
2. Slingshot
   1. Angle
   2. Power
   3. Stone Generation
   4. Testing
3. Health
   1. Generate health bar
   2. Health Depletion After hit
   3. Testing
4. Goliath
   1. Movement (Left/Right)
   2. Generation
   3. Hit Detection w/ Stone
   4. Testing
5. Design
   1. David
   2. Goliath
   3. Slingshot
   4. Stones
   5. Healthbar
   6. Title Screen
   7. Testing
6. Sound effects
   1. Fire
   2. Miss
   3. Hit
   4. Goliath Death
   5. Testing
7. Game Over
   1. Health Bar Depletion
   2. Testing

Froggy Jump Core Mechanics

1. Player
   1. Movement (Left/Right)
   2. Testing
2. Jumping
   1. Upwards velocity
   2. Gravity
   3. Testing
3. Screen
   1. Movement upwards with Player
   2. Screen does not move downwards
   3. Testing
4. Platforms
   1. Spawn Generation
   2. Recycling
   3. Hit detection for landing
   4. Testing
5. Spider hazards
   1. Spawn Generation
   2. Hit detection w/ Player
   3. Testing
6. Score tracking
   1. Visual indicator
   2. Track height of player from bottom
   3. Testing
7. Game over
   1. Hitting bottom of screen ends run
   2. Restart Timers
   3. Restart Score
   4. Restart Player
   5. Testing
8. Sprite switching
   1. Idle
   2. Jump
   3. Fall
   4. Testing
9. Sound effects
   1. Jump
   2. Hurt
   3. Die
   4. Testing

Duck Hunt – Craig

1. Shooting
   1. Mouse control
   2. Clicking/Shooting
   3. Testing
2. Ducks
   1. Movement
   2. Random spawn
   3. Hit Detection from Shots
   4. Testing
3. Game System
   1. Timer or Round System
   2. Scoring Update
   3. Testing
4. Sound effects
   1. Gunshot
   2. Hit
   3. Miss
   4. Testing
5. Sprite switching
   1. Flying Duck
   2. Hit Duck
   3. Falling Duck
   4. Character
   5. Testing
6. Game Over
   1. Completion of Rounds or Timer
   2. Display Score
   3. Reset Game
   4. Testing

**GUI System**

|  |  |  |
| --- | --- | --- |
| **Component** | **Task** | **Details** |
| Menu | Display Title Screen | Create main menu interface |
|  | Display Game Covers | Show game selection options |
|  | Testing | Verify menu functionality |
| Game Display | Navigation to Selected | Implement game launch system |
|  | Navigation Back to Menu | Create return mechanism |
|  | Testing | Test navigation flow |

**Tower Climb**

**Core Systems**

|  |  |  |
| --- | --- | --- |
| **System** | **Task** | **Details** |
| Platform/Terrain | Sprite Creation | Design platform graphics |
|  | Design Layout | Create level architecture |
|  | Testing | Verify terrain collision |
| Lives | 3 Lives System | Implement life counter |
|  | Hit Removes Life | Create damage system |
|  | Testing | Test life mechanics |
| Timer | Count Game Time | Track elapsed time |
|  | Testing | Verify timer accuracy |

**Player Mechanics**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Task** | **Details** |
| Player | Movement | Left/right/jump controls |
|  | Shooting | Projectile system |
|  | Power-up Collection | Item interaction |
|  | Door Interactions | Level progression |
|  | Testing | Test all player actions |

**Enemy System**

|  |  |  |
| --- | --- | --- |
| **Component** | **Task** | **Details** |
| Enemy | Spawning | Create enemy generation |
|  | Movement | AI behavior patterns |
|  | Attacks | Combat mechanics |
|  | Enemy Types | Different enemy variants |
|  | Testing | Test enemy behavior |

**Level Design**

|  |  |  |
| --- | --- | --- |
| **Element** | **Task** | **Details** |
| Floor | Level Layouts | Design each floor |
|  | Enter/Exit Gates | Transition mechanics |
|  | Enemy Composition | Balance enemy placement |
|  | Testing | Test level progression |
| Game Over | Beat Floor 10 | Victory condition |
|  | Lose All Lives | Defeat condition |
|  | Restart Game | Reset functionality |
|  | Testing | Test end conditions |

**Audio/Visual Effects**

|  |  |  |
| --- | --- | --- |
| **Category** | **Task** | **Details** |
| Sprite Switching | Move Animations | Player/enemy movement |
|  | Hit Animations | Damage feedback |
|  | Attack Animations | Combat visuals |
|  | Testing | Test animation triggers |
| Sound Effects | Damage Sounds | Player/enemy hurt audio |
|  | Attack Sounds | Combat audio |
|  | Door Sounds | Open/close audio |
|  | Testing | Test audio triggers |

**David vs Goliath**

**Character Systems**

|  |  |  |
| --- | --- | --- |
| **Character** | **Task** | **Details** |
| David | Player Positioning | Set up player character |
|  | Testing | Test player mechanics |
| Goliath | Movement (Left/Right) | Enemy movement pattern |
|  | Generation | Create Goliath character |
|  | Hit Detection | Collision with stones |
|  | Testing | Test enemy behavior |

**Combat Mechanics**

|  |  |  |
| --- | --- | --- |
| **System** | **Task** | **Details** |
| Slingshot | Angle Control | Aiming mechanism |
|  | Power Control | Shot strength system |
|  | Stone Generation | Projectile creation |
|  | Testing | Test shooting mechanics |
| Health | Health Bar Generation | Visual health display |
|  | Health Depletion | Damage system |
|  | Testing | Test health mechanics |

**Art & Audio**

|  |  |  |
| --- | --- | --- |
| **Category** | **Task** | **Details** |
| Design | David Sprite | Player character art |
|  | Goliath Sprite | Enemy character art |
|  | Slingshot Graphics | Weapon visuals |
|  | Stone Graphics | Projectile art |
|  | Health Bar UI | Interface elements |
|  | Title Screen | Main menu design |
|  | Testing | Test visual elements |
| Sound Effects | Fire Sound | Slingshot audio |
|  | Miss Sound | Failed shot audio |
|  | Hit Sound | Successful hit audio |
|  | Goliath Death | Victory audio |
|  | Testing | Test audio system |

**Game Flow**

|  |  |  |
| --- | --- | --- |
| **System** | **Task** | **Details** |
| Game Over | Health Depletion | Defeat condition |
|  | Testing | Test end game |

**Froggy Jump**

**Core Mechanics**

|  |  |  |
| --- | --- | --- |
| **System** | **Task** | **Details** |
| Player | Movement (Left/Right) | Horizontal controls |
|  | Testing | Test player controls |
| Jumping | Upward Velocity | Jump mechanics |
|  | Gravity System | Physics implementation |
|  | Testing | Test jump physics |
| Score Tracking | Visual Indicator | Score display |
|  | Height Tracking | Distance measurement |
|  | Testing | Test scoring system |

**World Systems**

|  |  |  |
| --- | --- | --- |
| **Component** | **Task** | **Details** |
| Screen | Upward Movement | Camera follows player |
|  | No Downward Movement | Prevent camera descent |
|  | Testing | Test camera system |
| Platforms | Spawn Generation | Create platforms |
|  | Recycling System | Reuse platform objects |
|  | Hit Detection | Landing mechanics |
|  | Testing | Test platform system |
| Spider Hazards | Spawn Generation | Create enemies |
|  | Hit Detection | Collision with player |
|  | Testing | Test hazard system |

**Game States**

|  |  |  |
| --- | --- | --- |
| **State** | **Task** | **Details** |
| Game Over | Bottom Screen Hit | Defeat condition |
|  | Reset Timers | Clear game timers |
|  | Reset Score | Clear score counter |
|  | Reset Player | Return to start position |
|  | Testing | Test reset functionality |
| Sprite Switching | Idle Animation | Stationary player |
|  | Jump Animation | Upward movement |
|  | Fall Animation | Downward movement |
|  | Testing | Test animation states |
| Sound Effects | Jump Sound | Movement audio |
|  | Hurt Sound | Damage audio |
|  | Die Sound | Game over audio |
|  | Testing | Test audio system |

**Duck Hunt**

**Shooting System**

|  |  |  |
| --- | --- | --- |
| **Component** | **Task** | **Details** |
| Shooting | Mouse Control | Cursor movement |
|  | Click/Shoot Mechanic | Fire system |
|  | Testing | Test shooting controls |

**Duck Mechanics**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Task** | **Details** |
| Ducks | Movement Pattern | Flight behavior |
|  | Random Spawn | Duck generation |
|  | Hit Detection | Shot collision |
|  | Testing | Test duck behavior |

**Game Systems**

|  |  |  |
| --- | --- | --- |
| **System** | **Task** | **Details** |
| Game System | Timer/Round System | Game progression |
|  | Score Updates | Point tracking |
|  | Testing | Test game flow |
| Game Over | Round/Timer Completion | End conditions |
|  | Display Score | Final score screen |
|  | Reset Game | Restart functionality |
|  | Testing | Test end game flow |

**Audio/Visual**

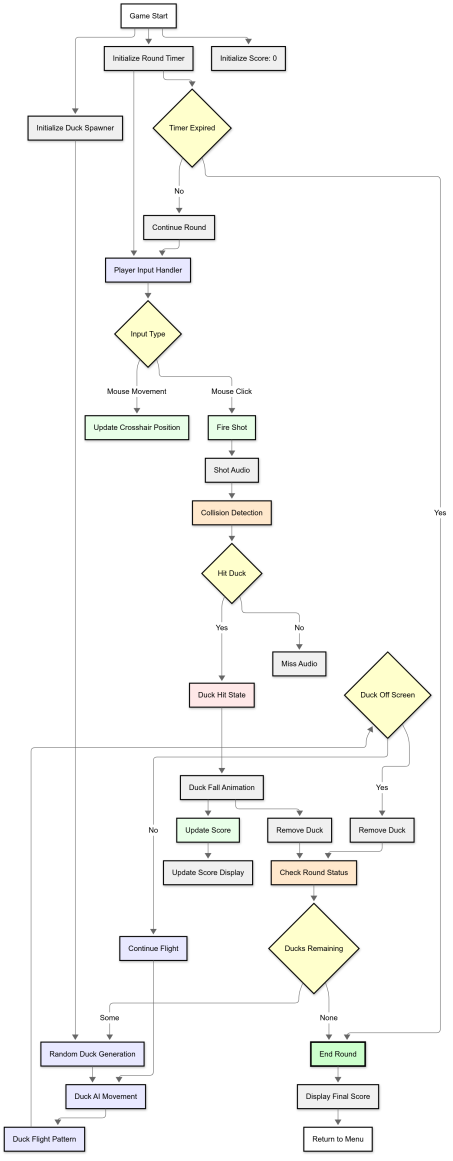
|  |  |  |
| --- | --- | --- |
| **Category** | **Task** | **Details** |
| Sound Effects | Gunshot Audio | Firing sound |
|  | Hit Audio | Successful shot |
|  | Miss Audio | Failed shot |
|  | Testing | Test audio triggers |
| Sprite Switching | Flying Duck | Normal state |
|  | Hit Duck | Damage state |
|  | Falling Duck | Defeated state |
|  | Character Sprites | Player graphics |
|  | Testing | Test animation system |

PROCESS FLOW DIAGRAMS:

Process Flow Description:  
This process flow chart illustrates the overall gameplay workflow, starting with player initialization and setup, then branching into player inputs, enemy behavior, and environmental interactions. Key processes such as movement, shooting, collisions, and power-ups drive the progression, leading to outcomes like advancing floors, respawning, victory, or game over.

A diagram of a company

AI-generated content may be incorrect.

A diagram of a company

AI-generated content may be incorrect.

A diagram of a company

AI-generated content may be incorrect.

PROJECT SCHEDULE:

# MONITORING & REPORTING:

For the purposes of communications, collaboration, and scheduling, we will be using myriad mechanisms. For communication, Discord, Zoom, and our Ivy Tech email and inbox depending on what type of communication it is. Zoom has been the easiest method to use. The downside to Zoom is the thirty-minute limit they put on meetings when using the free application of the tool. Discord has been a close second, in terms of ease of use and convenience. There have been microphone and connection problems, but we have been able to work around them. In addition, Discord offers a text stream where files and videos can be shared by the group. This is a great asset when trying to build a single product with group of people who don’t live close to one another. Discord also makes it easy to leave messages for the team at any hour. Everyone on the team can see them, and there is a record of what was said, by whom, and when. The other mechanism we have found to be particularly valuable is Trello. While GitHub is our main mechanism for collaboration, Trello makes arranging and organizing the tasks ahead a simple endeavor. Everyone can see what everyone is doing or has done, and there is a clear and prioritized list of what’s left to complete.

# APPENDIX:

# BitBox Arcade Project Activity Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Activity # | Description | Duration (Weeks) | Dependencies |
| **PLANNING** |  |  |  |
| 1.1 | Forms | 8 | None |
| 1.2 | Meetings | 8 | None |
| **HOSTING** |  |  |  |
| 2.1 | Local | 4 | 1.1, 1.2 |
| 2.2 | Windows and MacOS | 6 | 2.1 |
| 2.3 | Download | 2 | 2.2 |
| 2.4 | Testing (Hosting) | 2 | 2.3 |
| **GUI** |  |  |  |
| 3.1 | Menu | 3 | 2.1 |
| 3.2 | Game Display | 4 | 3.1 |
| 3.3 | Testing (GUI) | 2 | 3.2 |
| **TOWER CLIMB GAME** |  |  |  |
| 4.1 | Platform/Terrain | 2 | 3.2 |
| 4.2 | Player | 2 | 4.1 |
| 4.3 | Enemy | 2 | 4.2 |
| 4.4 | Lives | 1 | 4.3 |
| 4.5 | Timer | 1 | 4.3 |
| 4.6 | Floor | 1 | 4.3 |
| 4.7 | Game Over | 1 | 4.4, 4.5, 4.6 |
| 4.8 | Sprite Switching | 2 | 4.7 |
| 4.9 | Sound Effects | 2 | 4.7 |
| 4.10 | Testing (Tower Climb) | 2 | 4.8, 4.9 |
| **DAVID VS GOLIATH GAME** |  |  |  |
| 5.1 | David | 2 | 3.2 |
| 5.2 | Slingshot | 2 | 5.1 |
| 5.3 | Health | 1 | 5.2 |
| 5.4 | Goliath | 2 | 5.2 |
| 5.5 | Design | 3 | 5.3, 5.4 |
| 5.6 | Sound Effects | 2 | 5.5 |
| 5.7 | Game Over | 1 | 5.6 |
| 5.8 | Testing (David vs Goliath) | 2 | 5.7 |
| **FROGGY JUMP GAME** |  |  |  |
| 6.1 | Player | 2 | 3.2 |
| 6.2 | Jumping | 2 | 6.1 |
| 6.3 | Screen | 2 | 6.1 |
| 6.4 | Platforms | 2 | 6.2, 6.3 |
| 6.5 | Spider Hazards | 2 | 6.4 |
| 6.6 | Score Tracking | 1 | 6.5 |
| 6.7 | Game Over | 1 | 6.6 |
| 6.8 | Sprite Switching | 2 | 6.7 |
| 6.9 | Sound Effects | 2 | 6.7 |
| 6.10 | Testing (Froggy Jump) | 2 | 6.8, 6.9 |
| **DUCK HUNT GAME** |  |  |  |
| 7.1 | Shooting | 2 | 3.2 |
| 7.2 | Ducks | 2 | 7.1 |
| 7.3 | Game System | 3 | 7.2 |
| 7.4 | Sound Effects | 2 | 7.3 |
| 7.5 | Sprite Switching | 2 | 7.3 |
| 7.6 | Game Over | 1 | 7.4, 7.5 |
| 7.7 | Testing (Duck Hunt) | 2 | 7.6 |

## Summary Statistics

* **Total Activities:** 39
* **Estimated Project Duration:** 18-20 weeks
* **Critical Path:** Planning → Hosting → GUI → Game Development → Testing
* **Testing Phases:** 6 dedicated testing periods ensuring quality at each major milestone