Production II Project

Technical Specifications

(Jak Tiano, Ian Sartwell, Ryan Leslie, Shain Strother, Evan Schipellite, Tim House)

Prepared by: Evan Schipellite

Introduction / Platform Specifications

The entire duration for this project, including conceptual creation, will span approximately twelve iteration cycles. The purpose of this document is not specifically intended to detail the iteration tasks, as that particular content will be included within a Milestone Guide Document. Instead, this document is meant to reflect the overall platform specifications that will be established during the final phases of the project, as well as detail the design patterns and structures that will be conducted when creating the features for the game. In this sense, this document should primarily be reviewed by the Programming team, as well as any inquiring members, in order to further understand the methods and procedures that will be utilized to tackle and resolve the creation of the game's core mechanics.

The game project will be developed using the Unity Game Engine, and the InControl Input System, provided by Patrick Hogan, will be included as an external library to resolve conflicts relating to Unity's process of reading and writing controller information. Otherwise, the game's code structure will be entirely built in C# using Unity to establish the view for the gameplay. The eventual release of the game is intended initially for sale on PC, and Marketing will be responsible for contacting and communicating with potential retailers. In that sense, the game will primarily be built and available as a downloadable game type, therefore stress tests will be conducted to ensure that the content size of the game is kept to a minimum and the actual gameplay is able to run effectively on both Mac / PC. Gameplay should also be playable on a variety of computer specifications, within reason.

The nature of the requirement for controllers, however, would potentially limit the player audience due to the lack of reliability that consumers may have all controllers available for gameplay. In that light, while the initial platform and release will be primarily directed toward the PC market, the game will also be developed and released on console markets, primarily the Xbox Live Arcade and the Ouya Market. Knowledge of the code structures and game requirements for both markets is crucial to eventually taking the project as is and redefining its directions to meet documented expectations for both markets. Due to the code's intended and flexible structure, the addition of various aspects such as Leaderboards, should be easily implemented in order to meet publishing requirements. Therefore, while PC is the initial platform developed on, further development of the Vertical Slice would lead to collaboration

with Marketing to effectively port and adjust the project to meet requirements for various console markets.

Project Specifications

The game is intended to be a local multiplayer combat game for two to four players. Furthermore, the game requires players to utilize controllers and share a single screen for the duration of gameplay. With this in mind, the next aspect of this document serves to review several of the features required for the game in order to detail the actual specifications for development. Risk analysis is conducted in a separate Technical Risk Assessment Document, and should be reviewed at a later time by the Programming team to evaluate the complexity behind each particular feature or groups of features. It is also important to note that, due to the iteration nature of the project, all features listed and discussed are subject to design changes, and therefore the document should reflect the current status and understood direction of the entirety of the project cycle. For simplicity purposes, the following features will be broken down into general groups and further detailed in order to better relate to the Technical Risk Assessment Document.

Local Multiplayer

The game will only need to be playable with a minimum of two players, and a maximum of four. This removes the possibility of allowing a gameplay experience in a single player setting. Still, due to the nature of local multiplayer, various features will need to be incorporated and tested over the duration of this project. These mainly relate to controller functionality, as well as connections and disconnections.

<u>Controller Layout</u>

- Left-Axis
    - Controls player movement
    - Movement is absolute
        - If possible, players will move in the direction requested
        - A range of around the possible directions will filter input
            - IE: A directional request will also take into account angles +/- X around the possible directions.
- Right-Axis
    - Controls arm direction
    - Direction is absolute
        - Arms will instantly snap to desired direction

    - Orientation flipping should not alter current arm direction
- Start Button
    - Pauses the Game
    - Potentially brings up menu for game options
        - This may be restricted to the host (Player 1)
- Left Trigger
    - Activates Leg ability
- Left Back Trigger
    - Activate player jump
    - Also activates toggle gravity
        - The mechanic will be detailed later
        - When aerial, player can press in order to attempt gravity swap
- Right Trigger
    - Activate Torso ability
- Right Back Trigger
    - Activates Arm ability

Standardized Input

Due to the reality of varying controller types and formats, it is important that the game is able to run effectively utilizing a number of primary controller types. Xbox and PS3 controllers should all function with gameplay. Therefore, the implementation of an external library providing Standardized Inputs should be researched further. This procedure will enable the game to detect connected devices, select Device Profiles, and consistently monitor device status during gameplay. While the majority of the functionality will be handled by the InputManager, it is still important that the Programming team understands the specifications regarding the accessibility and use of the Input Manager.

The InControl system will provide the game with accessibility to an Input Manager that can provide information relating to devices at runtime. This will allow the access and detection of connections and disconnections during gameplay. Therefore, during menu screens and combat, the game will be able to adapt to changes regarding controller status. Upon connection, the game should seek to relate the next incremental player without a device. Upon disconnection, the game should seek to remove the corresponding player's device and prepare them for a reconnection if applicable. During the selection process, this detection of controllers will be used to specify the current players entering the game. During gameplay, this will be only used to keep track of player inputs and properly reconnect controllers upon loss. Therefore, in-game disconnected players will simply remain idle, while during the selection screen disconnection will server to remove disconnected players.

It should further be noted that the InControl system simplifies button and axis detection. Due to the nature of detecting absolute movement for both the player movement and arm

direction, the InControl system not only provides that ability to detect button presses, but it also features a method to acquire vector information from an entire access. This information can be utilized to map the vector information to the world space in order to compare the requested direction with the possible direction and rotation the player mechanics can take. This should simplify some of the vector math discussed later on in this documentation.

Menu System

While simplified, the menu system should provide an accessible manner for players to begin, exit, and start games utilizing the first player's controller to select game options and start methods to begin new rounds. Once in the selection screen, other controllers will be able to connect to the current game in order to begin selecting robot parts, enabling a ready status when they are prepared for gameplay. Therefore, the following section serves to briefly define the various menu requirements for the project, along with the details relating to each specific feature.

- Main Menu
    - Conveys the Start, Exit, and Options menu
    - Possible art directions may call for parallax backgrounds or gameplay activity in background, however this is not currently required for the menu procedures
    - Player 1 should be able to navigate vertically through menu with Left-Axis
    - X (Primary button) should enable selection of an options element
    - Exit game shut down the current session
- Options Menu
    - Similar navigation style to the Main Menu
    - O (Back button) allows for retreat to previous menu
    - Offers sound volumes
        - Effects
        - Music
    - Offers credit information
        - Brief list of developers on project
- Start Game
    - Leads to Inventory Selection
- Inventory Selection Menu
    - Broken up into four windows
    - Each window represents a player
    - Upon connection and Primary Button, players can connect to the game
    - Upon disconnection or Back Button, players can be removed from the game
    - Left-Axis-Y navigates through a vertical menu representing the robot structure
        - Torso
        - Arms

        - Legs
    - Left-Axis-X navigates through a horizontal menu representing part types
    - An individual vertical column may allow the selection of a player's color
        - These will likely be defined colors
        - Upon selection, the color becomes unobtainable by other players
    - By pressing the Right Trigger, players can ready for gameplay
    - When all players are ready, the game begins after a brief delay
    - If all players remove themselves, the game returns to the Main Menu
- Game Over Menu
    - At the end of the game, a winner is displayed
    - Game technical information may be conveyed below each player box
        - Player structure
            - Shows selected Torso, Legs, Arms
        - Player Passive Aspects
            - Health
            - Movement
            - Jump
        - Damage Done
        - Players Finished
    - Player 1 can then navigate with the Left-Axis-Y to select an option
        - Restart Game
        - Return to Inventory Selection
        - Return to Main Menu
    - Primary Button selections option
        - Holds until all player's ready-up
            - Right Trigger selects ready for non-Player 1
            - Cannot be undone
- Pause Menu
    - All players can pause
        - Start Button toggles pause
    - Player 1 has option Game Over Menu options
        - Restart Game
        - Return to Inventory Selection
        - Return to Main Menu

Player Architecture and Mechanic Creation

Representing the core aspect of the game, the player architecture should be created using polymorphic aspects to represent the player parts. Ideally, a player base class will be created to

handle all similar features including health, movement, jump power, and player robot creation. It may be worth splitting the base into a movement class and part class to link similar features. The Movement class will handle the actual force understanding for player movement, jumping, and adherence to forces. The Base class will handle the creation of the robot, the value checks, and the monitoring of player actions. Each part will be derived from a default part. Torso, Legs, Arms, and Projectiles will all be derived from base classes that unite similar traits that are apparent in all similar part types. Due to the nature of C#, this aspect should not be difficult to implement and will allow for the extensibility of part features, actions, and types, as well as allow parts to dictate health, movement, and jump increases or decreases that can be relayed to the Base class.

Player Movement

- Handles player movement, gravity forces, and jump.
- Movement
    - Determines the speed in which the player moves around planets when on the ground
    - On Ground Checks
        - Ray casts downwards from a 'bottom' point
        - Bottom point manually placed within Leg prefab
        - Detects non-projectile object below within distance
            - TransformDirection → Allows for down vector to be oriented with player status
    - Ground movement
        - The player will be translated / rotated around planets when on the ground
        - Absolute Movement
            - Compare input vector / required vector
                - Allow for angle +/-
            - If correct, move player left / right
                - Translate / rotate around the planet
    - Air movement
        - Add forces to player to limit air movement
        - Player can still move left and right
        - Forces are applied in direction
            - Max force before landing
            - IE: Players can only adjust their air movement a specific amount when in the air
                - This limits players fighting against other forces while still letting them move around if no other forces are exerted
    - Jump
        - On button press, the player can apply a large amount of force upward

        - While not on the ground, this action can be executed
            - A short delay is implemented to prevent the user from applying the force multiple times due to the nature of raycasting
                - IE: The speed of frames would detect 4-6 jump presses, thus applying the force 4-6 times before the user actual jumped high enough to leave the ray cast range
        - Jumping again while attempt an orbit toggle
            - Player will queue the GravityObject class
                - This class will attempt an orbit swap if applicable
- Gravity influence
    - Each update, the player will also queue the GravityObject class for a gravity source
        - This will apply a subtle force toward the active gravity source
        - This allows the players to be attracted to planets and other gravity objects

<u>Player Base</u>

- Handles player health, and part models / scripts
- Player Values
    - After initializing player parts
        - Evaluate the total player health
            - Add together part health
        - Evaluate the total player movement speed
            - Take base movement
            - Add part movement
                - These may be either positive / negative
            - Game will check for below 0
                - This should never happen
                - Will set to 1 regardless
            - Send player movement speed to Movement class
        - Evaluate Player Name
        - Evaluate Player Color
- Create Player Robot
    - 3 Different types of enums: Torso, Legs, Arms
        - Previous gameplay will set enum types for each player
    - Reflection Pattern allows for the enums to create models / scripts
        - IE: Mech_Legs_Spider enum creates the corresponding model and script
    - Initialization creates Torso, Arm, and Legs before initializing player values
    - Each part starts as default, if a non-default enum is detected
        - Remove the default model
        - Add the corresponding part model

            - LoadResources(PATH + ENUM)
        - Add the corresponding script
            - mechArmScript = GetComponent<ENUM>()
- Player Mech Base will update and monitor device connections / disconnections
- Arm Direction
    - Evaluates vector input of Right-Axis to orientate player arms
        - Transforms vector to world space
- Actions
    - Runs the actions on the scripts for Torso, Arms, and Legs
    - Each script will check for actual button presses

Mechanic Features

The following section serves to outline the details following each part type. Essentially, the Base class will call the actual actions for each part script. These scripts will update accordingly and further detect for button presses in their own ways. Each part will derive itself from the Torso, Legs, or Arms class. All classes will share similar features relating to their methods for cooldown components, health, movement adjustments, etc. All of these variables will be held by the parent class, and the child classes will implement additional required variables that may depict aspects such as damage, resources to load, and specified cooldown times. These variables will be readily available and named properly for further testing and adjustments by the Design team.

- Legs
    - <u>Tank Treads</u>
        - Collision with Tank Treads deals passive damage over time to enemies
        - Active button rotates Tank Treads 360 degrees, stretching them outward and damaging enemies caught within range
        - This may also temporarily disable nearby enemies, allowing the player to pursue or flee the area
        - This effect can likely be generated through simple collision checks and scaling of the view model
    - <u>Jetpack</u>
        - The Jetpack should allow the player to hover while aerial
            - Applied forces upwards
        - The player will reduce their downward fall
        - The player can move around within the air for a short time
        - The player can damage enemies below their Jetpack
            - Creation of invisible collision objects that last a short time below the player

- Spider
  - Allows player to dive toward the ground
  - Impact with object or ground triggers explosion
    - Invisible collision object that lasts a short duration
  - Players are damaged on impact
  - Players are damaged on explosion
  - Players maintain constant force while used
- Torso
  - Body Slam
    - Allows player to dash forward
      - Applies force to player
      - Creates collision object around player
        - Damages enemies hit
        - Applies force to enemies on hit
          - Determine direction between collision object and enemy hit to acquire force direction
  - Laser Beam
    - Activates laser bombardment that spans in front of player
    - Fires multiple laser shots
      - Trigger colliders
      - Can pass through enemies
      - Still destroyed on non-player / non-projectile collision
    - Shots deal damage depending on range
      - Close range deals more
  - Energy Shield
    - Allows player to damage and knock-back other players
      - Creates visual sphere that sends enemies flying away
        - Decreased damage / force depending on distance from center of shield
      - Lasts a short duration
      - Reflects enemy projectiles
        - Re-orients the current force on the projectile
- Arms
  - Sword
    - Player can swing sword to damage players in front
    - Sword is treated as a trigger object
      - Upon collision, knock-backs collision object and damages
    - Sword undergoes rotation process
      - On activate, sword rotates for a set amount
      - At end of swing, sword rotates backwards to beginning

- On collision, sword rotates backwards to beginning

- Fists of Fury
  - Each button press causes player to punch in arm direction
    - Extends arm length for punch
    - Damages enemies hit
    - Damaged enemies remembered by fists
      - Each hit refreshes remember duration
      - After remember duration reaches 0
        - Removes enemy from known list
      - Each hit checks for remembered enemies
        - Additional damage applied
          - Depends on times hit before
  - Possible subtle knock-back on hit
      - Used to interrupt enemies
- Machine Gun
  - Allows player to quickly use burst-fire from a distance
    - Over duration fires a certain number of shots
      - Shots vary slightly in horizontal velocity
        - Causes shot spread
  - Minor damage on hit, many shots fired
  - Somewhat influenced by gravity forces
- Rocket Launcher
  - Allows player to fire individual rockets
  - Rockets extend with an initial velocity
    - Gravity causes rockets to fall back toward planet
    - Homing encourages rockets to move toward enemies
      - Doesn't follow owner
  - Explosion on object hit
    - Damages enemies
    - Reduced damage to owner
- Riot Shield
  - Allows player to 'punch' in the arm direction
    - Shield moves forward
    - Damages players hit
    - Knocks players back
  - On active, reflects projectiles and prevents weapon hits
    - Reflected projectiles are converted to Riot Shield player
    - Weapon hits (Melee) may do reduced damage
- Grapple Whip
  - Launches a grapple whip to pull enemies closer

        - Arm animates and extends outwards
        - Players hit are disabled
        - Arm pulls players hit back to the Grapple Whip player
        - Damages players hit
- Projectiles
    - All projectile will derive from a base Projectile class
    - Projectile will share initial force, damage, and impact force to apply
    - Projectile will also be able to toggle a homing feature to follow players
        - This feature can have variable detection distance and % adjustment
    - Projectile also have a Strength value and Strength damage
        - Determines the projectile health and damage it deals to other projectiles
        - This allows projectiles to destroy other projectiles
            - IE:
                - Rocket has 10 health, 1 strength
                - Pellet has 1 health, 1 strength
                    - Rocket requires multiple pellets to be destroyed, but can easily destroy other weaker projectiles before then

Physics System

The Physics system is best understood as being divided into two basic areas. There are Gravity Sources and Gravity Objects. Gravity Sources serve as objects that attempt to attract Gravity Objects to their centers. Gravity Sources don't actually seek to apply their force, but rather are available for Gravity Objects to check for and queue for forces. In this sense, Players and Projectiles can check for collisions with Gravity Source colliders (Orbits) and queue the source for its force.

Gravity Sources

As noted beforehand, objects with a Gravity Source scripts will simply maintain their position and be prepared to supply it to inquiring Gravity Objects.

Gravity Objects

Gravity Objects will begin with an empty list of Gravity Sources. Upon Trigger Enter / Exit, the Gravity Object will remove to add or remove the Gravity Source as applicable. The first Gravity Source in the list will be utilized for gravity applications. Therefore, the first Gravity Source located will remain as the primary gravity application. However, if a Gravity Object, such as the player, calls the swap function, the Gravity Object will attempt to alter the primary Gravity Source.

In this case, the Gravity Object will do one of three things. If there is one or less Gravity Sources, nothing will occur. If two Gravity Sources exist in the known list, the Gravity Object will swap their locations, thus changing the primary Gravity Source. If more than two Gravity Sources exist, the Gravity Object will attempt to evaluate the nearest secondary Gravity Source and then swap the two. This will allow objects, such as Players, to remain attracted to specific planets, thus enabling them to jump without concern for leaving their current planet. Player control will also allow them to choose when to toggle to switch to a new orbit within range. Furthermore, if the Gravity Object Exits all triggers, it will maintain a separate variable to determine the last know Gravity Source in order to continue a gravity application.

UI Elements

While largely depicted on the Design and Art task list, the functionality of the UI elements and external aspect of gameplay will initially stem from functionality creation by the Programming team. Simple elements such as Camera functions, Health display, and Cooldown display represent the core UI elements that the programmers will need to create before other teams begin testing and implementing potential designs.

- Camera
    - Camera should lerp between spots
        - Doesn't instantly snap for viewing
    - Camera should attempt to follow all viewable players
        - Add or removes players dynamically
            - Player death removes
            - Player creation adds
        - Takes average of player locations to determine position and zoom
    - Camera clamp
        - Detects objects (Players) that go beyond a certain clamp range
            - Camera views the object position beyond this range at a max
            - IE: Player that goes past a distance of 10 will be understood as being at 10
            - Visual elements can be placed at these max ranges to point to the direction the player is actually at
    - Offers various values
        - Min Zoom
        - Max Zoom
        - Camera Lerp Speed
        - Camera Height
- Health Display

    - Displays a health bar above the player
        - Orientates with player
        - Conveys the percentage health remaining
- Cooldown Display
    - May undergo several redesigns
    - Conveys the cooldown time for Torso, Legs, and Arms
    - Each corner of the screen shows the robot structure in the player color
        - Slightly transparent
        - Components turn red of use, clear up as they become available
        - Light up on available
        - Potentially show on viewer model when ready

Stretch Goals

The following seeks to simply detail some of the technical requirements behind available stretch goals for the current project cycle. The Technical Risk Assessment document will further elaborate on the reasons behind the scoping of these features, but this section intends on mainly noting the actual required technical specifications for each feature, thus providing the Technical Risk Assessment with basis for evaluation.

- First Phase Gameplay
    - Allows players to compete for parts
        - This would limit their selection in the combat gameplay
        - Designed to allow players to experience new part mixtures
        - Places a combination of strategy of part collection and combination
    - Players use the Left-Axis only for this gameplay
    - Players move around in a 2D space, chasing after parts that spawn
    - Planets are represented by 2D cut-outs
        - Originally voxel drilling was intended, but this is the simplified version
    - Players move through planets
    - Parts spawn over the duration time limit
        - Spawn in multiple groups
        - Calculations can dictate the relative balance of part types
    - A single balanced competition level can be created by Designers
    - Colliding with another player may simply knock-both players in different directions
        - Collisions from the back or sides may disable that player for a moment
    - At the end of the round, players move to the Inventory Selection Menu
- Player Eject on Death
    - Upon loss of health, players leave their Robot as a small person
    - Similar control scheme

        - Jump
        - Torso
            - Roll
        - Arms
            - Punch or Pistol
        - Legs
            - Kick
    - Minimum damage
    - Player has small amount of health
    - Player can attempt to survive and pick-off other players still fighting
- Player Profiles
    - Players can create new profiles from the Main Menu
        - Defines
            - Player Name
            - Color
            - Statistics
                - Wins / Losses
                - Highest Damage in a round
    - Profiles can be selected from a list in the Inventory Selection Menu