Technical Specification

Puppeteer

**Group Members:**

Sandra Andersson **Dev**

Carl Appelkvist **Dev**

Ludvig Björk Förare **Dev**  
Anton Jonsson **Dev**

Kristoffer Lundgren **Dev**  
Filip Rehnman **Dev**

Philip Stenmark **Dev**

Benjamin Vesterlund **Dev**

Oscar Leiner Olsson **Artist**

Gustav Mårdestam **Artist**

Robert Ringholm **Artist**

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Name** | **Comment** |
| A | 2019-04-11 | Anton Jonsson  Philip Stenmark  Kristoffer Lundgren | Milestone 2 |

# Game Mechanics

## Platform and OS

Windows 7 or higher (64-bit only)

i7 - 6700k @ 4.00GHz or AMD equivalent

Nvidia Geforce GTX-980 Ti or AMD equivalent

16 GB RAM

## External Code

Unity 2018

Oculus SDK for VR-support

Mirror by vis2k for networking

FMOD for sound.

## Game Objects

### Level

**Components**

* LevelBuilder

Attributes

* + GameObject[] Modules
  + GameObject StartModule
  + GameObject GoalModule
  + int NumModules

Methods

* + Start

### Room

Room should be synced with Mirror

**Tag**

* Connectable

**Components**

* CooldownComponent

Attributes

* float CooldownTime
* bool Available

Methods

* get/set Available
* StartCooldown
* Mesh Renderer
* Box Collider

### Puppet

**Tag**

* Player

**Components**

* HealthComponent

Attributes

* + [SyncVar] int Health
  + int MaxHealth
  + int MaxRegenHealth
  + int RegenSpeed
  + int RegenCooldown

Methods

* + Damage
  + Heal
  + Update
* Interactable (acts as base class for all in-level interactable items)
* DF

Methods

* OnInteractBegin
* OnInteractEnd
* OnRaycastEnter
* OnRaycastExit
* Image (UI image for compass)
* PowerUpBase (acts as base class for all power-ups)

Attributes

* + int Duration

Methods

* + OnActivate
  + OnComplete
* StaminaPower : PowerUpBase

Attributes

* float SpeedModifier
* InvisibilityPower : PowerUpBase
* NavigationPower : PowerUpBase
* DetectionPower : PowerUpBase
* PlayerController : Interactable

Attributes

* float MovementSpeed
* float LookAhead
* int Ammunition
* bool HasMedkit
* bool PowerupReady
* GameObject CurrentWeapon
* float ReviveTime

Methods

* Update
* Use
* Mesh Renderer
* Rigidbody
* Capsule Collider (For collision detection)
* Animator

### Puppeteer

**Components**

* Mesh Renderer
* GrabTool

Attributes

* + GameObject SelectedObject
  + float SnapDistance

Methods

* + Update
  + Pickup
  + Drop
  + CanConnect
  + FindNearestAnchor

### Enemy

**Tag**

* Enemy

**Components**

* HealthComponent

Attributes

* + int Health
  + int MaxHealth
  + int MaxRegenHealth
  + int RegenSpeed
  + int RegenCooldown

Methods

* + Damage
  + Heal
  + Update
* PathFinderComponent

Attributes

* + List CurrentPath
  + float MovementSpeed

Methods

* + MoveTo
  + Update
* StateMachineComponent/EnemyController

Attributes

* State CurrentState
* GameObject EnemySpawner
* GameObject TargetEntity
* float AttackCooldown
* int AttackDamage

Methods

* SetState
* Update

States (classes)

* AttackingState

Methods

* Enter
* Run
* Exit
* ReturnToSpawnerState

Methods

* Enter
* Run
* Exit
* WanderState

Methods

* Enter
* Run
* Exit
* SeekState

Methods

* Enter
* Run
* Exit
* Mesh Renderer
* Rigidbody
* Capsule Collider
* Animator

### EnemySpawner

**Tag**

* EnemySpawner

**Components**

* EnemySpawner

Attributes

* float SpawnRate
* GameObject EnemyPrefab

Methods

* Update
* Mesh Renderer
* HealthComponent

Attributes

* int Health
* int MaxHealth
* int MaxRegenHealth
* int RegenSpeed
* int RegenCooldown

Methods

* Damage
* Heal
* Update
* Box Collider

### Weapon

**Tag**

* Weapon

**Components**

* WeaponComponent : Interactable

Attributes

* int Capacity
* Int Ammunition
* float FiringSpeed
* int Damage
* float Spread
* float Recoil
* float ReloadTime
* int NumberOfShots

Methods

* Use
* Reload
* Mesh Renderer

### Trap

**Tag**

* Trap

**Components**

* TrapComponent

Attributes

* int Damage
* float ActivateTime

Methods

* OnTriggerEnter
* OnTriggerStay
* Box Collider
* HealthComponent

Attributes

* int Health
* int MaxHealth
* int MaxRegenHealth
* int RegenSpeed
* int RegenCooldown

Methods

* Damage
* Heal
* Update
* Mesh Renderer
* Animator

### Door

**Tag**

* Door

**Components**

* DoorComponent : Interactable

Attribute

* bool open
* bool Locked

Methods

* Update
* Mesh Renderer
* Box Collider (For not being able to walk through the door)
* Box Collider (Used as trigger for proximity)

### Compass

**Components**

* CompassComponent

Attribute

* GameObject[]
* Vector3 north

Methods

* AddObject
* RemoveObject
* Update

### PowerUpRecharge

**Components**

* MeshRenderer
* Recharge/Refill : Interactable (used to refill the player’s power)
* BoxCollider

### Medkit

**Components**

* Medkit : Interactable

Attribute

* HealAmount

## Control Loop

Unity handles everything that has to do with the control loop.

## Data Flow

Unity handles the storing and loading of all files and assets. Every component written by the dev team should aim to load everything that is going to be used frequently into memory during the loading phase in the unity Start() and Awake() functions.

## Game Physics and Statistics

### LevelBuilder

The LevelBuilder Start() method should only happen on the server. When spawning a new room as the host, you need to specify that with Mirror.

**Start**

All room modules that may be included in the level generation are provided in the **Modules** game object array with an exception of the start and goal rooms, which are provided separately as **StartModule** and **GoalModule**. Once the LevelBuilder is started, the level is generated in its entirety. It is however required that the start and goal rooms are reachable from each other.

### HealthComponent

The Health attribute should be synced with Mirror, meaning all methods changing health (Heal & Damage must happen on the server)

**Damage**

Damage is a method used for removing health from the **Health** attribute in **HealthComponent**. It takes the amount of health to lose as a parameter. This function is called from either the **WeaponComponent** on the player or from the enemy if it does damage to a player. If the health reaches zero upon taking damage, some action need to be handled, i.e. the player downing, spawner exploding or enemy dying. This can be determined by the HealthComponent by comparing the tag on its parent and calling different methods depending on the outcome.

**Heal**

Heal is a method used for adding health to the **Health** attribute. It takes the amount of health to add as a parameter. Heal is responsible for making sure the Health attribute never goes above the **MaxHealth** attribute.

**Update**

Update is responsible for Regeneration of health using all **Regen** attributes. If the is no generation, only the **Heal** function will affect the health positively.

### PlayerController

The player must be synced with Mirror. Internal variables such as ammo and movementSpeed does not unless implementing anti-cheat.

**Update**

Handles player movement using inputs. Limits velocity according to the **MovementSpeed**. The update function also raycasts straight ahead from the center of the player view to identify interactable items. The player always tracks the latest looked-at object.

**Use**

Attempts to perform a Use action. Depending on the looked at object, an appropriate event is selected, such as picking up an item, recharging power up or opening a door. This may be done using various tags for all interactable items.

**OnInteractBegin**

The interactor attempts to start reviving the player if below zero health. The function also verifies that the interactor has a medkit to perform this action. The revive action is not instant, but instead uses the **ReviveTime** attribute.

**OnInteractEnd**

The interactor will stop the revive if not done already.

### PowerUpBase

PowerUpBase might need a bool or something to sync in order to for example make Gekko invisible for Puppeteer but not puppets.

**OnActivate**

Override this function to specify behavior when the power up is activated. The activation will start the power up usage countdown using the **Duration**. The OnActivate function may be implemented as a coroutine to simplify logic.

**OnComplete**

Override this function to specify behavior when the power up is suspended. The completion function may for example remove the Pekko’s goal icon from the compass.

### GrabTool

**Update**

Checks all inputs relevant to the grabbing and dropping mechanic of the puppeteer and calls the relevant methods depending on the output. Also updates the currently selected room and placement guide depending on mouse/VR hover.

**Pickup**

Pickup works with both items such as spawners and traps, and also rooms. For rooms specific rules need to be followed: Pickup picks up the selected room from the puppeteers perspective. The pickup function creates a copy of the selected object that can be lifted from the ground. The source object is however still in place until a new position is fully selected. Furthermore, a guidance object is also created from a lightweight clone of the source object that acts as an outline and views the most optimal placement option.

**Drop**

Drop has to do different checks depending on if you are holding a room or just a trap/spawner. For a room, it drops the picked up room into the most optimal position in the level layout, if any. When a room is dropped, the selected object clone and its guide object is destroyed. If a dropped room does not fulfill all rules for connecting (using the CanConnect function), it is discarded.

**CanConnect**

Returns a bool telling whether or not a room can be dropped in a certain place. The function should handle all placement rules. A source anchor point and destination anchor point should be provided. To modify the maximum snapping distance between anchors, the **SnapDistance** property is used. The connection rules are:

* + - Both anchors must be valid.
    - Anchors must be open for connection (not already used).
    - The forward vector of the anchors must be opposite of each other.
      * Vector3.Dot(srcForwardNormalized, dstForwardNormalized) == -1
    - The distance between the anchors must be less than the **SnapDistance**.
      * Vector3.Distance(srcPosition, dstPosition) < SnapDistance

**FindNearestAnchor**

Locates the nearest anchor (door) to the selected object, if any.

### CooldownComponent

**get/set**

A public getter and a private setter for the attribute Available. When a room is move the StartCooldown function is called and the Available attribute is set to false.

**StartCooldown**

Starts the cooldown timer of the room and compares it with the CooldownTime attribute. When the cooldown is over the attribute Available is set to true.

### PathfinderComponent

**MoveTo**

Pathfinds to coordinates specified in parameter and updates **CurrentPath** attribute.

**Update**

Follows the **CurrentPath** attribute by adding force vectors along the path on the rigidbody. If there is no current path do nothing.

### StateMachineComponent

**SetState**

Changes state.

**Update**

Runs the current states **Run** function.

### EnemySpawner

Needs to be synced with Mirror

**Update**

Spawns an **EnemyPrefab** instance at some nearby position in a predefined range of the spawner position if the spawn rate timer reaches zero. The **SpawnRate** variable determines how often the update function may perform its logic. May also use a coroutine for a simpler implementation.

### WeaponComponent

Must be synced with Mirror.

**Use**

Attempts to fire the weapon. The function may immediately fail if there is no ammunition left or if the time since last weapon use does not exceed the firing speed. The weapon may also not be fired while reloading. In case the firing was successful, the following steps are performed:

* For each bullet in the total number of simultaneous bullet fired:
  + Calculate bullet spread using the **Spread** variable
  + Raycast directly from the viewport center into the direction of the camera plus the angle deviation from the spread
    - ViewportPointToRay(new Vector3(0.5f, 0.5f, 0.0f))
  + Collect the raycast hit information and fetch its **HealthComponent**
  + If there is a health component, deal some damage using the **Damage** variable
  + Create bullet impact effect or other effect
* Play appropriate firing animation and effects
* Perform recoil physics on the weapon
* Play appropriate weapon firing sound
* Reset firing speed timer
* Decrease ammunition count

**Reload**

Reloads the weapon using the ammunition the weapon owner is carrying. The function attempts to fill the weapon to its maximum **Capacity**, if possible. If the weapon is fully loaded, its **Ammunition** will equal its **Capacity**. Some mechanism may also be required to prevent firing the weapon while loading. A reload sound and animation is also played.

**OnInteractBegin**

The interactor picks up this weapon. If the interactor already has a weapon component attached, the new weapon is automatically swapped for the new one. Some logic is here required to properly attach transforms between weapon and interactor.

### TrapComponent

Must be synced with Mirror.

**OnTriggerEnter**

Detects when a game object is within the trap activation area. The function immediately triggers the **ActivateTime** countdown. If the countdown reaches zero, the trap is activated and is therefore considered consumed.

**OnTriggerStay**

If the activation countdown reaches zero, the game object that is staying within the trigger area is dealt some **Damage** if the target has a **HealthComponent**.

### DoorComponent

Must be synced with Mirror.

**Update**

Checks if the door should close.

**OnInteractBegin**

Toggles the door from the open and closed positions. The door may not be opened when the attribute **Locked** is set to true. The rotation of the door is determined by the position of the interactor, meaning it may rotate both inwards and outwards.

### Interactable

**OnInteractBegin**

Override this function in the subclass for handling interaction events. This callback also provides a parameter **GameObject** that represents the interacting game object (interactor).

**OnInteractEnd**

Override this function in the subclass for handling end of interaction events. This callback also provides a parameter **GameObject** that represents the interacting game object (interactor).

**OnRaycastEnter**

Enables the outline of the object.

**OnRaycastExit**

Disables the outline of the object.

### CompassComponent

**AddObject**

Registers a new ‘trackable’ entity to the compass, e.g. teammates or level objects. Accepts any GameObject as parameter as long as it contains some Icon component.

**RemoveObject**

Unregisters a game object from the list.

**Update**

Updates all relative positions of the registered objects in the compass user interface. The icon of each object is displayed in the compass UI element. The compass is configured using some predefined ‘north’ direction. Some nitty gritty maths is here required to map 3D direction vectors to the 2D user interface element.

### PowerupRecharge

Must be synced with Mirror.

**OnInteractBegin**

Refills the player’s power if applicable.

### MedkitComponent

Must be synced with Mirror.

**OnInteractBegin**

Adds the medkit to the player’s inventory if the player doesn’t have one.

## Artificial Intelligence

The enemies are controlled by a simple Finite State Machine(FSM):

States

* AttackingState
* ReturnToSpawnerState
* WanderState
* SeekState

**AttackingState**

The enemies enter the attacking state when they have a clear line of sight of the player and they are within range. While in this state the enemies follow the player and automatically attack them when within range.

**ReturnToSpawnerState**

When the spawner that spawned an enemy is attacked all the spawned enemies will enter this state and return to the spawner in order to protect it. When a spawner or an enemy is moved by the puppeteer this behavior is lost.

**WanderState**

When an enemy doesn’t see any players it will wander on its own to nearby rooms to search for players.

**SeekState**

This state will pathfind to a players last known location or back toward spawn if it has run to far from home.

## Multiplayer

Mirror by vis2k solves all of our multiplayer issues. The data is sent using TCP and a client-host system.

# User interface

## Game Shell

* **Splash screen**

The first screen you see when you open the game. After any button is pressed, the menu should be opened.

* **Main menu**

The main menu is just a way to get to the other menus.

* **Video options**

Video options is where you can change resolution and enable/disable fullscreen.

* **Audio options**

Audio options should implement sound sliders for all types of sound in the game.

* **Controls**

In controls you should be able to rebind all keys used in the game and also change the sensitivity of the mouse.

* **Join**

In Join you enter an IP-address to connect to a host and enter a lobby

* **Lobby**

In a lobby you choose character and indicate when you are ready to start the game

* **Host/Lobby**

The same as ordinary lobby but for the host of the game.

## Main Play Screens

### Puppet

* Health from the HealthComponent displayed as a number.
* Stamina from the PlayerController displayed as a bar.
* Ammunition from PlayerController displayer as a number.
* Power-up status displayed as a bar.
* Compass that points towards other game objects using their transforms to calculate angles.
  + Also displays cardinal directions relative to some fixed point.
  + Uses unique icons to represent different game objects, such as players.
* Ammunition and Capacity from the equipped weapon as numbers

### Puppeteer

* Cooldown on rooms from CooldownComponent
* Room selection highlights calculated by the GrabTool
  + Uses some outline shader to display optimal room placement and the selected room’s previous position.

# Art and Video

Unity handles loading, storing and drawing of the art. The animation transitions are done using an Animator Controller.

# Graphics Engine

Unity handles all graphics rendering in the game.

# Artist Instructions

Use file formats supported by Unity. Otherwise they have free hands.

# Sound and Music

Unity has built in support for all the standard audio formats. Sounds are played by an Audio Source and they are picked up by an audio listener which is normally placed on the player’s camera. 3D sound, filters, and drop-off are all controlled inside the audio source.

## Sound Engineering Instructions

Use file formats supported by Unity. Otherwise they have free hands. FMOD will be used.

# Level Specific Code

The level is randomly generated at the start of the game to get a different layout each time. This is done using the LevelBuilder component in the Level game object. It is required that the start and goal rooms always are reachable after each level generation.