Technical Design Document

Game Details

- **Game Name:** A Lizard’s Tail
- **Team Name**: Lizards are wizards

Team Members

Name	**Role**	**Responsibilities**
Brendan Blue	Designer	Lead narrative design
Dris Hunt	Designer	Lead systems design
Chris Selleck	Designer	Lead level design
Martin Widdowson	Artist	Lead rigger
Brea Fox	Artist	Lead environment artist
Charles Spall	Artist	Lead effects artist
Elise Allan	Artist	Lead character artist
David Flintoft	**Programmer**	Lead programmer

Game Concept

The game is a 2D puzzle platformer that involves controlling a Lizard Wizard through a level using various movement mechanics and magical spells. The player can move, jump, crawl, and climb up walls. They are also able to cast spells to clear paths, defeat slimes, or for use in puzzle solving.

Technical Goals

Technical Goal 1 - Fair & Fun 2D Player

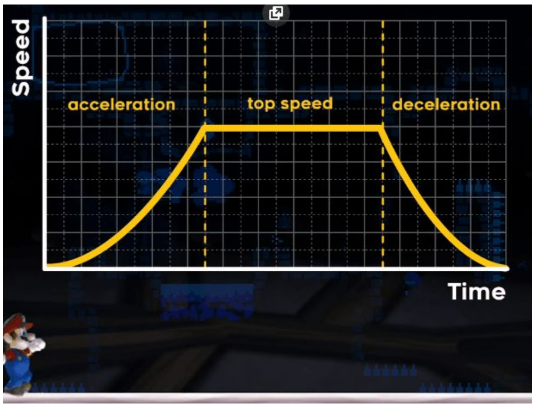
Who's responsible: David.

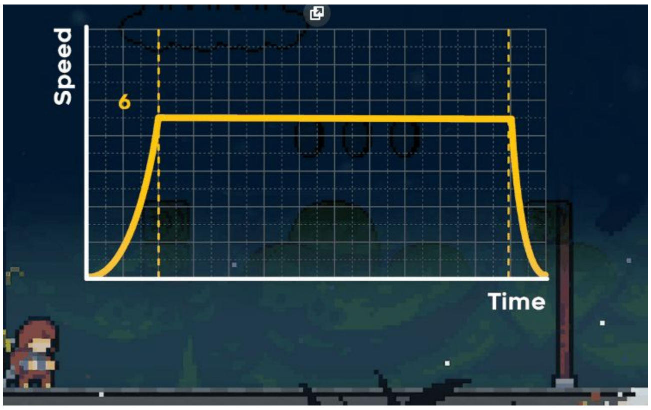
Description: It is crucial that a 2D platformer feels fun and fair for the user to play. Various features must be implemented to achieve a balance of fun but not too forgiving. This can be realised through various 2D techniques, listed below:

Responsive controls

When the player switches direction whilst on the ground it should be very quick. When the player is in the air, switching directions should be more floaty. There is also how fast the player should reach maximum speed and how quickly the player should come to a stop. Fiddling with these values can achieve movement that feels good to play. Different games use different values as shown in the images below.

The YouTube channel, Game Maker's Toolkit, has delved into this core technique in their video covering Celeste (2018): https://youtu.be/yorTG9at90g?t=94. The reason the game feels so good is due to the incredibly responsive, yet lenient controls. If the character doesn't feel good to control due to factors such as floaty jumps, bad physics or buggy collisions, then the game is not going to be good. The designers can change the values to decide how responsive the game is.

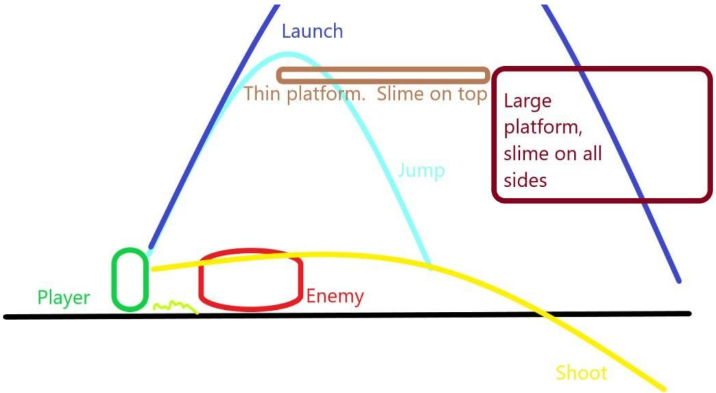




Jump height and variable jump height

Deciding on a jump height for the player that isn't too floaty but allows enough time to position the landing is tricky. How this is implemented depends on the game and what the game is aiming for.

For our puzzle platformer game the lizard will need enough height to jump over the larger slimes and up onto platforms. Below is a concept for the scale of the jump with regards to other objects (light blue line).

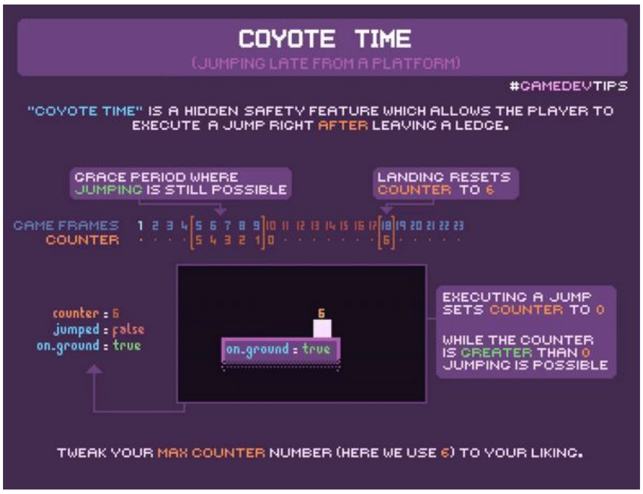

Launch
Thin platform. Slime on top
Large platform, slime on all sides
Jump
Player
Enemy
Shoot

Using variable jump height gives the user more control over how their player moves and is used in majority of platformers. This is achieved typically by performing a full-height jump when the user presses the jump button, then artificially increasing gravity if the user lets go of the button.

```
if ( canJump && WasButtonPressed(Jump) ):
      Jump()
elif ( !IsButtonHeld(Jump) && velocity.y > 0 ):
      ApplyDownwardsForce()
```

Coyote time

This is a form of input buffering that still allows the player to jump if they press the jump button too late. It is important because it makes the game feel more fair and doesn't expect the player to always make precise inputs. Also, inputs from controllers may be delayed or a monitor may be slightly behind.



COYOTE TIME
(JUMPING LATE FROM A PLATFORM)

#GAMEDEVTIPS

"COYOTE TIME" IS A HIDDEN SAFETY FEATURE WHICH ALLOWS THE PLAYER TO EXECUTE A JUMP RIGHT AFTER LEAVING A LEDGE.

GRACE PERIOD WHERE JUMPING IS STILL POSSIBLE

LANDING RESETS COUNTER TO 6

EXECUTING A JUMP SETS COUNTER TO 0

WHILE THE COUNTER IS GREATER THAN 0 JUMPING IS POSSIBLE

TWEAK YOUR MAX COUNTER NUMBER (HERE WE USE 6) TO YOUR LIKING.

GAME FRAMES 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
COUNTER · · · · 5 4 3 2 1 0 · · · · · · · · 6 · · · · ·
counter : 6
jumped : false
on.ground : true
6
on.ground : true

GIF Reference: https://twitter.com/Case_Portman/status/1178342795033092097

“Inverse” coyote time

This is also a form of input buffering but instead will allow the player to press the jump button too early before landing, but still execute a jump. Below is a snippet of pseudocode of how this might be implemented using simple timers.

```
inverseCoyoteDelay = 0.02
inverseCoyoteTimer = 0.0

if ( Jump is pressed && not grounded ):
      inverseCoyoteTimer = inverseCoyoteDelay

elif ( inverseCoyoteTimer > 0 )
      inverseCoyoteTimer -= deltaTime

...

if ( inverseCoyoteTimer > 0 && grounded )
      Jump()
      inverseCoyoteTimer = 0.0
```

Lenient collisions

It is also important for 2D platformers to have lenient collisions. Friendly objects should have larger hitboxes so the player can more easily interact with them. Whereas dangerous objects (enemies, spikes, etc.) should have smaller hitboxes so it is harder for the player to hit it.

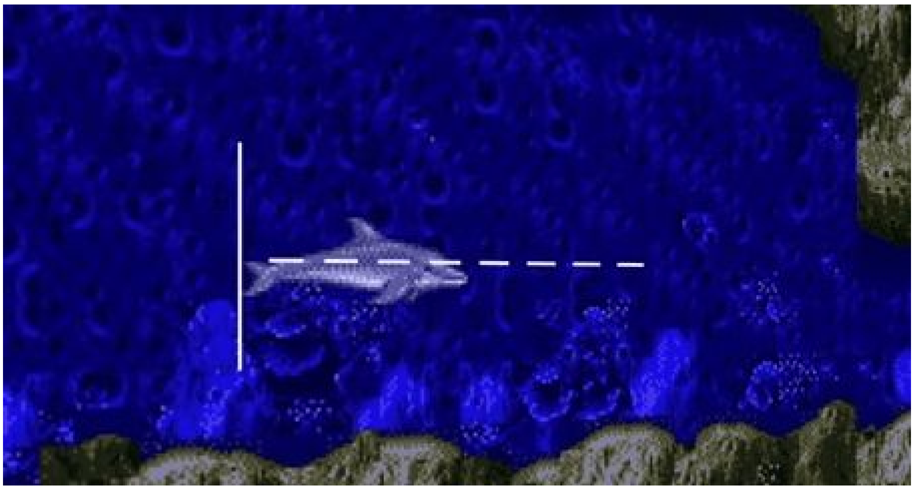
Technical Goal 2 - 2D Camera Controller

Who's responsible: David.

Description: Choosing what type of camera to use is important for a 2D game. There are many types, such as: look ahead, camera trap, lock-on, smooth follow, etc. Developing a custom 2D camera that can switch between these different types can allow more complex level design. Adding functions like screen shake and zoom to the game can make it feel more alive.

David will create a camera script that can do all of these types of cameras. Each mode is a different 'state' that the camera can be in - and the movement of the camera is determined by the state. This script can then be used in future 2D projects too.

For the game, we plan to use a look ahead camera. In this state, the view of the camera will be shifted to look ahead of the direction that the player is facing. When the player switches directions the camera will either snap or lerp to the other side.



Reference: https://youtu.be/l9G6MNhfV7M?t=308

Technical Risks

Technical Risk 1 - 2D in Unity

What's the risk about: Unity is a game engine originally designed for 3D games and due to this, is not the best choice for developing 2D games. However, over the years the Unity developers have added more and more 2D functionality. This is still restrictive, especially for platformers that require precise control over the physics and pixel-perfect collision detection. These issues require special consideration when thinking about how to create the game because the most important thing about a 2D platformer is the 2D platforming. If it doesn't feel good to play then it won't be fun for the user no matter how many cool mechanics there are.

How will the risk be mitigated: David will find custom 2D character controllers and test them to see which will be right for our game. This is challenging because every character controller works differently. Some use custom physics and collision detection, and some use Unity's physics. Finding one that feels good and is customisable enough for the game will prove to be a challenge.

Technical Risk 2 - Support for Multiple Controllers

What's the risk about: The game is being targeted for Windows PC and PS4. This requires compatibility for Mouse/Keyboard, Xbox360 Controller and PS4 Controller. The issue is that controller inputs vary between different types. For example, *joystick button 0* is *A* on an Xbox360 controller, but is *Square* on the PS4 Controller. This makes it difficult to set up inputs in Unity and is the only difficulty for producing on PS4.

How will the risk be mitigated: David will do research into how to use custom input managers that can standardise controller inputs. This will significantly simplify the code, reducing the need for checking what platform the game is running on. If he can't find one then a custom script will be made to make the inputs easier to manage - this may take more time.

Features/Mechanics/Tasks

Feature/Mechanic	**Who's responsible**	**Scheduled Date**
Basic player movement & platform collisions	David	Alpha, 21/10/19
2D camera & message box	David	Alpha, 21/10/19
Slime movement A.I.	David	Alpha, 22/10/19
Magic shards & player spell-casting	David	Alpha, 28/10/19
Player interactions with slime	David	Alpha, 29/10/19
Advanced player movement (climbing / crawling)	David	Alpha, 1/11/19
Health system	David	Beta, 1/11/19
Game UI & pause menu	David	Beta, 4/11/19
Checkpoints & respawn system	David	Beta, 4/11/19
Spikes & flammable doors	David	Beta, 4/11/19
Dragonfly power-up	David	Beta, 5/11/19
Collectable gems	David	Beta, 5/11/19
Exit level condition	David	Beta, 11/11/19
Main menu hub	David	Beta, 11/11/19

Deliverables

Deliverable	**Who's responsible**	**Format**
Game Trailer		H.264 MP4 with a minimum 1280x720 resolution (30 - 50 seconds)
Demo Video		H.264 MP4 with a minimum 1280x720 resolution (1:00 - 1:20)
PC Build / Installer	David	*Executable .exe*
PS4 Build	David	

System Requirements

Target Device 1 - Windows PC

Recommended hardware: 2D platformers generally don’t need high-end hardware to run smoothly. Many of these types of games on Steam recommend 1GB to 2GB of RAM.

Platform-specific requirements: None.

Target Device 2 - Playstation 4

Recommended hardware: The game should run perfectly fine on the base PS4 model.

Platform-specific requirements: PS4 requires the use of a PS4 controller which will need to be taken into consideration when dealing with user input.

Third Party Tools

- Unity Engine (Version: 2018.3.8f1)
- Microsoft Visual Studio 2017 (Version: 15.9.2)
- Perforce version control.
- Character Controller 2D (https://github.com/prime31/CharacterController2D)
- ProBuilder (Version 4.1.0)
- ProGrids (Version 3.0.3)

File Formats

Models:	.FBX, .ABC
Textures:	.PNG
Sounds:	.WAV

Coding Conventions

CSharpGuidelines (Version 5.4.0)

- https://csharpcodingguidelines.com/
- https://github.com/dennisdoomen/csharpguidelines/releases/tag/5.4.0

Source Control

Source control repository: Perforce.

Source control client tools: PV4, Unity.

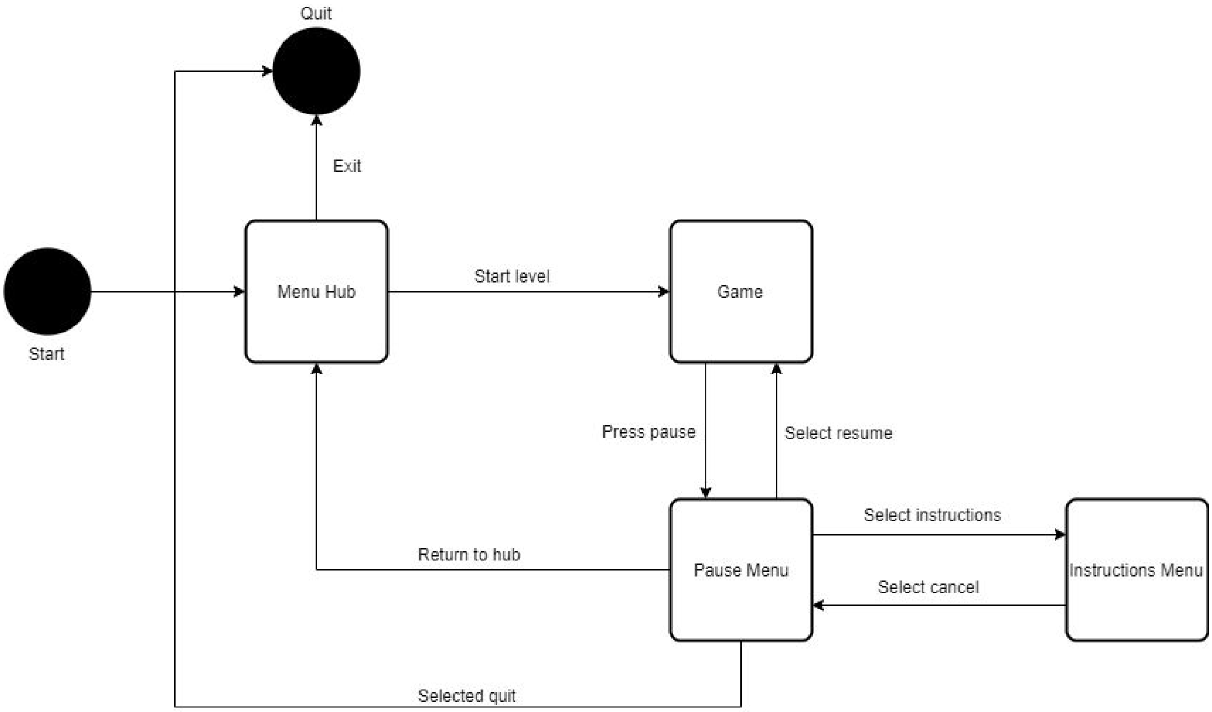
Commit message formats: The message should explain the changes that are being made and, if applicable, why the changes were made. It should also be signed off using a name.

Other repo notes: When working with source control and Unity, there are some folders that should be excluded from the syncing progress.
Using a branch of the repo can be used for testing or implementing new features.

\

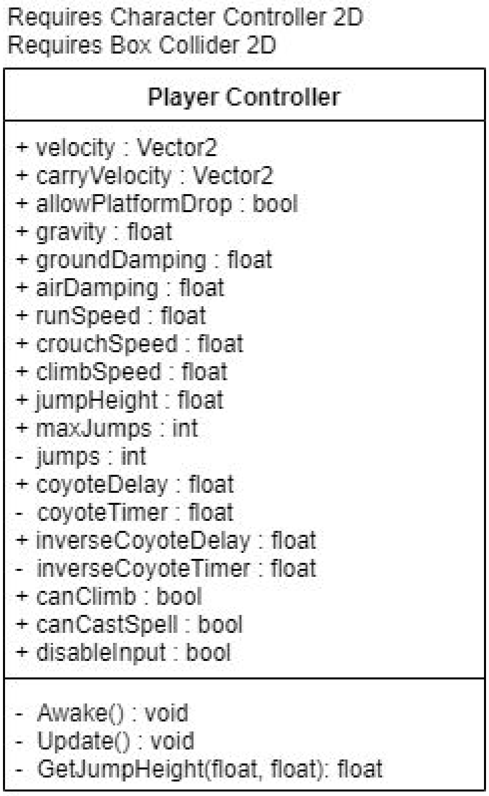
Game Flow

Scene	**Who's responsible**	**What it does**
Main Menu (Hub)	David	Playable hub area - starts a new game, or exits.
Instructions	David	Display instructions - accessed via the pause menu.
Game	David	Playable game level
Pause	David	Pause menu - accessed from in game.



Game Objects and Scripts

Player Controller



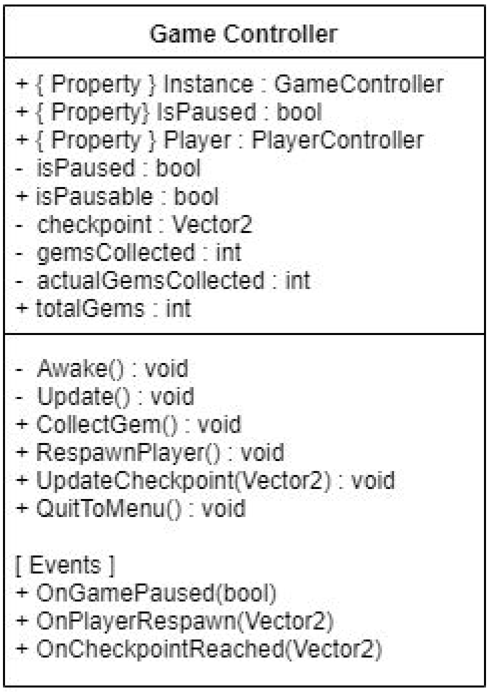
Requires Character Controller 2D
Requires Box Collider 2D

Player Controller

+ velocity : Vector2
+ carryVelocity : Vector2
+ allowPlatformDrop : bool
+ gravity : float
+ groundDamping : float
+ airDamping : float
+ runSpeed : float
+ crouchSpeed : float
+ climbSpeed : float
+ jumpHeight : float
+ maxJumps : int
- jumps : int
+ coyoteDelay : float
- coyoteTimer : float
+ inverseCoyoteDelay : float
- inverseCoyoteTimer : float
+ canClimb : bool
+ canCastSpell : bool
+ disableInput : bool

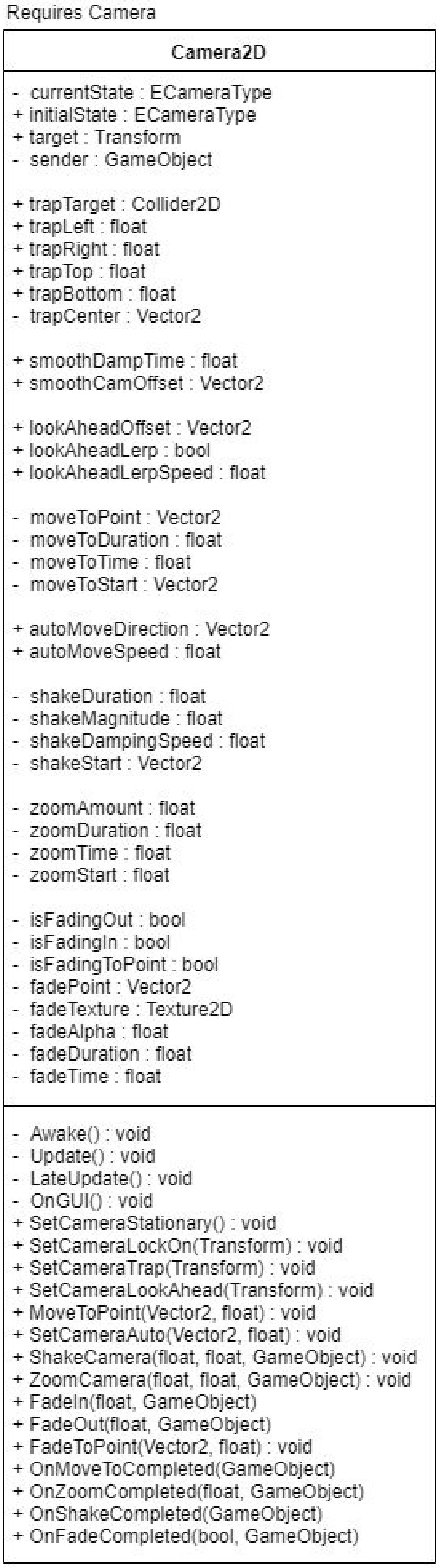
- Awake() : void
- Update() : void
- GetJumpHeight(float, float): float

Game Controller



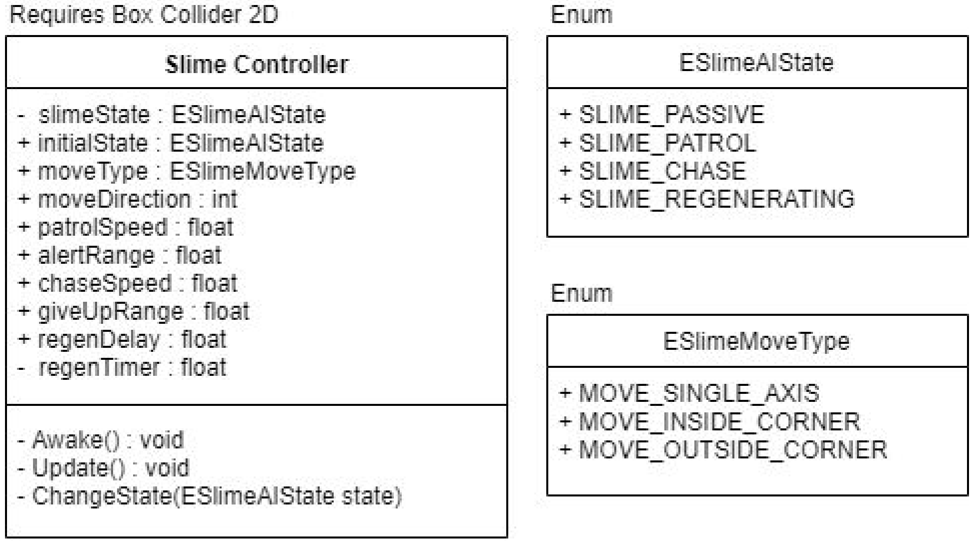
```
Game Controller

+ { Property } Instance : GameController
+ { Property} IsPaused : bool
+ { Property } Player : PlayerController
-  isPaused : bool
+ isPausable : bool
-  checkpoint : Vector2
-  gemsCollected : int
-  actualGemsCollected : int
+ totalGems : int

-  Awake() : void
-  Update() : void
+ CollectGem() : void
+ RespawnPlayer() : void
+ UpdateCheckpoint(Vector2) : void
+ QuitToMenu() : void

[ Events ]
+ OnGamePaused(bool)
+ OnPlayerRespawn(Vector2)
+ OnCheckpointReached(Vector2)
```

2D Camera



```
Requires Camera

Camera2D

-  currentState : ECameraType
+ initialState : ECameraType
+ target : Transform
-  sender : GameObject

+ trapTarget : Collider2D
+ trapLeft : float
+ trapRight : float
+ trapTop : float
+ trapBottom : float
-  trapCenter : Vector2

+ smoothDampTime : float
+ smoothCamOffset : Vector2

+ lookAheadOffset : Vector2
+ lookAheadLerp : bool
+ lookAheadLerpSpeed : float

-  moveToPoint : Vector2
-  moveToDuration : float
-  moveToTime : float
-  moveToStart : Vector2

+ autoMoveDirection : Vector2
+ autoMoveSpeed : float

-  shakeDuration : float
-  shakeMagnitude : float
-  shakeDampingSpeed : float
-  shakeStart : Vector2

-  zoomAmount : float
-  zoomDuration : float
-  zoomTime : float
-  zoomStart : float

-  isFadingOut : bool
-  isFadingIn : bool
-  isFadingToPoint : bool
-  fadePoint : Vector2
-  fadeTexture : Texture2D
-  fadeAlpha : float
-  fadeDuration : float
-  fadeTime : float

-  Awake() : void
-  Update() : void
-  LateUpdate() : void
-  OnGUI() : void
+ SetCameraStationary() : void
+ SetCameraLockOn(Transform) : void
+ SetCameraTrap(Transform) : void
+ SetCameraLookAhead(Transform) : void
+ MoveToPoint(Vector2, float) : void
+ SetCameraAuto(Vector2, float) : void
+ ShakeCamera(float, float, GameObject) : void
+ ZoomCamera(float, float, GameObject) : void
+ FadeIn(float, GameObject)
+ FadeOut(float, GameObject)
+ FadeToPoint(Vector2, float) : void
+ OnMoveToCompleted(GameObject)
+ OnZoomCompleted(float, GameObject)
+ OnShakeCompleted(GameObject)
+ OnFadeCompleted(bool, GameObject)
```

Slime



Requires Box Collider 2D

Slime Controller

- slimeState : ESlimeAIState
+ initialState : ESlimeAIState
+ moveType : ESlimeMoveType
+ moveDirection : int
+ patrolSpeed : float
+ alertRange : float
+ chaseSpeed : float
+ giveUpRange : float
+ regenDelay : float
- regenTimer : float

- Awake() : void
- Update() : void
- ChangeState(ESlimeAIState state)

Enum

ESlimeAIState

+ SLIME_PASSIVE
+ SLIME_PATROL
+ SLIME_CHASE
+ SLIME_REGENERATING

Enum

ESlimeMoveType

+ MOVE_SINGLE_AXIS
+ MOVE_INSIDE_CORNER
+ MOVE_OUTSIDE_CORNER

Gameplay Systems

Gameplay System 1 - Slime A.I.

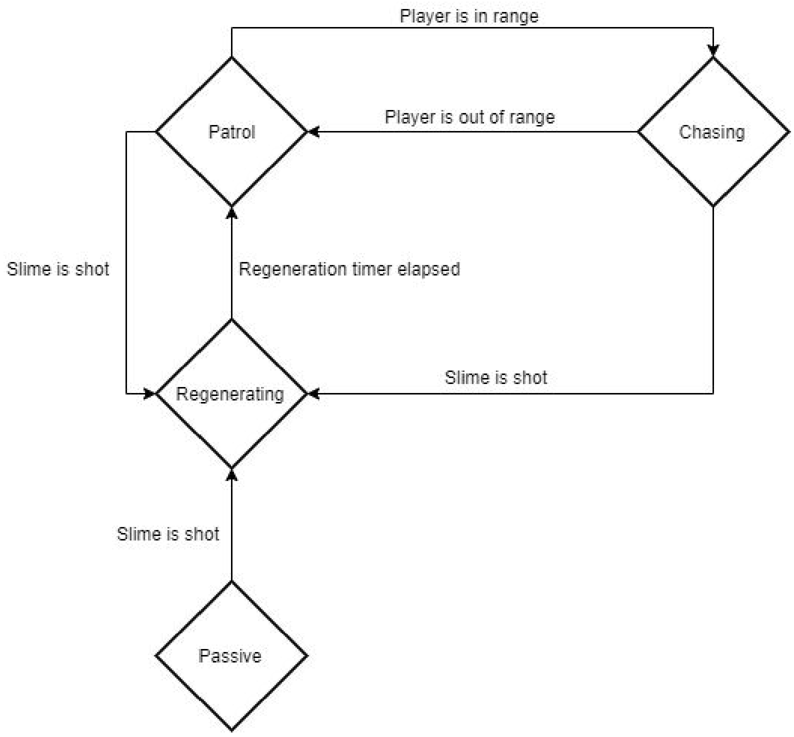
Who's responsible: David

Description: The slimes will have three observable A.I. behaviours: patrol, chase and regenerate. They will also have multiple methods of movements:

- Single axis: either horizontally or vertically. When the slime reaches the edge of the platform it will simply switch direction.
- Around corners: either inside or outside corners.

These various parameters must be visible for the designers to edit. This allows some slimes in the level to chase down the player whereas others might not. Having the multiple movement methods provides interesting gameplay situations.

When the slime is patrolling it moves back and forth until the player is within a certain range. It then switches to the chase behaviour where it will try to reach the player. The slime is unable to jump over platforms so it is possible that it may not be able to reach the player in a given situation. The state will revert back to patrolling if the player gets far enough away from the slime. However, if the player strikes the slime with a fireball spell it will enter the regenerating state. During this state the slime will slowly re-assemble itself. This means that slimes can never truly be destroyed, allowing back-tracking to still provide a challenge.



Gameplay System 2 - Game Controller

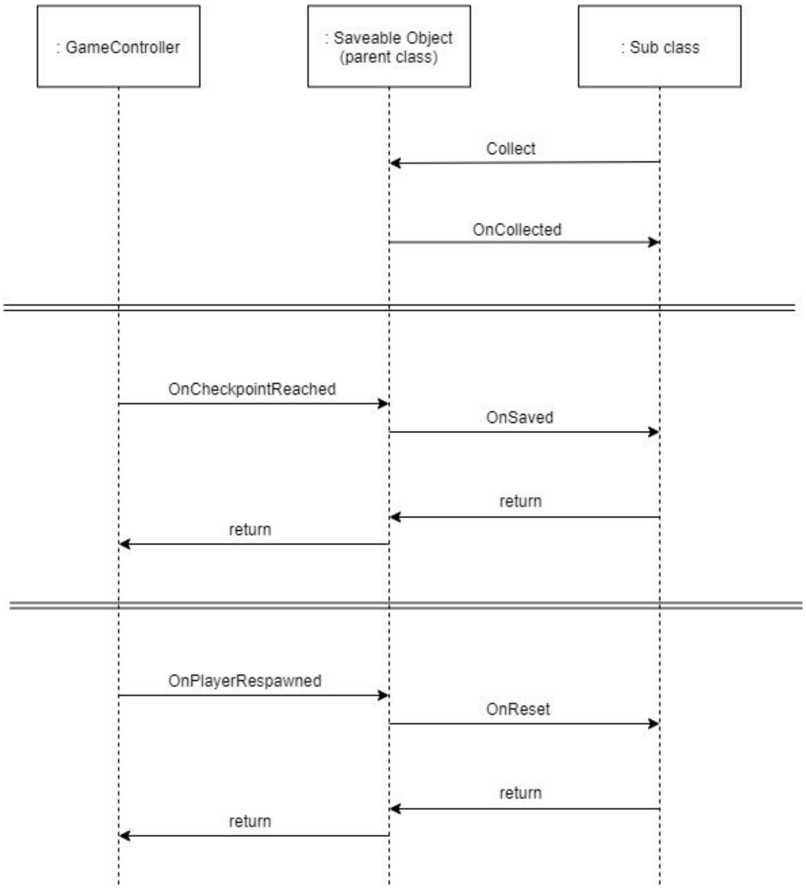
Who's responsible: David

Description: The Game Controller is a script that any game object will be able to access. It will be implemented as a singleton but will also need to be a component so it can utilise Unity's *OnAwake()* and *OnUpdate()* methods. The script keeps track of the number of gems, the checkpoints reached and whether the game is paused or not. Using the publisher-subscriber design pattern, game objects can be notified when certain events occur. For example: a gem that was collected by the player should respawn if the player dies before reaching a checkpoint. Once the player reaches a checkpoint, the gem is saved as 'collected' for the duration of the level, whether or not the player dies again. This system allows quick player respawns without having to load the entire scene over again.

The events available for game objects to subscribe to will be:

- OnGamePaused()
- OnPlayerRespawn()
- OnCheckpointReached()

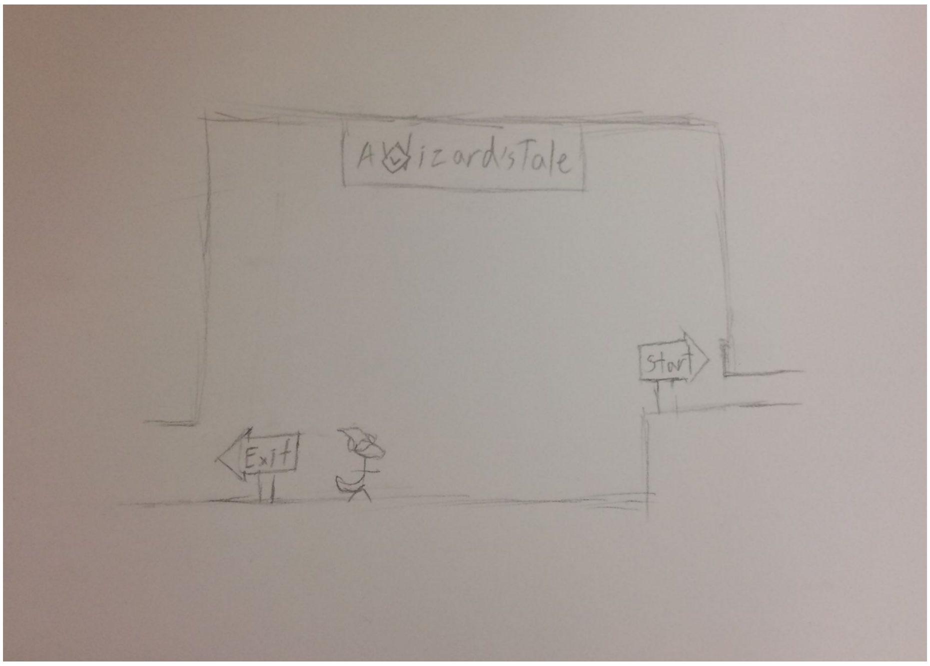
I will implement a *SaveableObject* abstract class that components can inherit from. This will allow an automated saving process instead of hard-coding each object. This class needs to interface with the Game Controller by subscribing to its events.



Input Method(s)

Target platform	Input system	Who is responsible
Windows PC	Mouse/Keyboard	David
Windows PC	Xbox360 Controller	David
PS4	Playstation 4 Controller	David

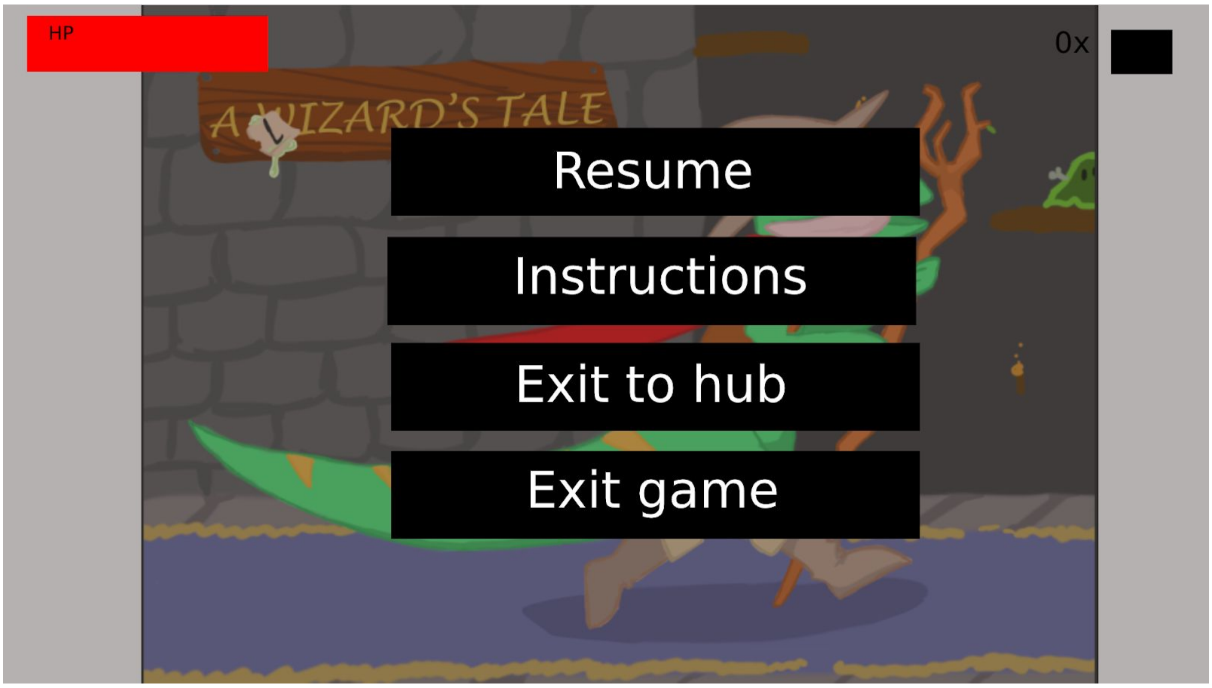
User Interface



Main Menu Hub Mock-Up (Chris)

The main menu is a hub level where the player can control the character. It will have a clear exit and start area. The instructions and credits will be displayed diegetically in the background of the level.

In-game UI (Charles)
The top-left will display the health of the player and the top-right will show the number of gems collected.



Pause menu (Charles)
The game will pause in the background and the user can select one of the four options on the menu. The health and gem counter are still visible, however in the pause menu the gem counter will also display how many total there are in the level. I.e. 10/100.