MOLEZ

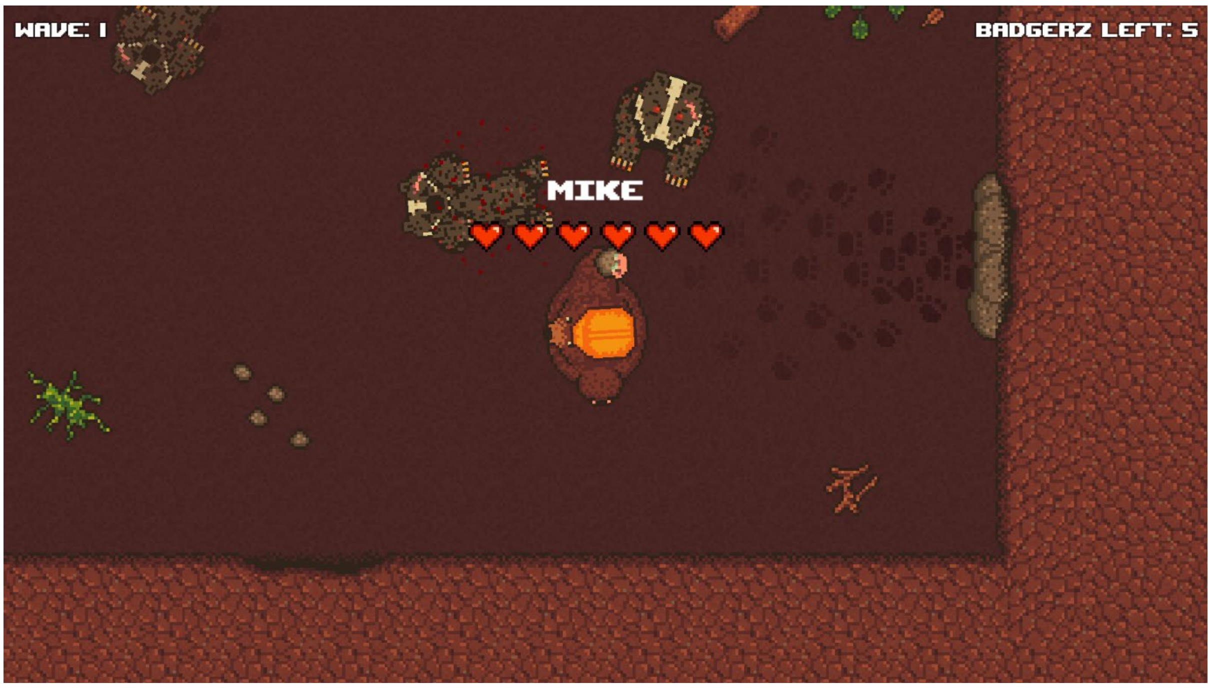
GAME DESIGN DOCUMENT

A MOLE-THEMED WAVE-BASED CO-OP SURVIVAL GAME

MIKE HOLBROOK

CONTENTS

OVERVIEW



SUMMARY

Set in a moles burrow, players must fight off waves of oncoming zombie badgers, completing mini-games to unlock doors and revive players. The badgers are hungry for moles, and they are relentless in their hunt for blood.

MoleZ is a twin stick wave survival game inspired by games such as Call of Duty: Zombies and Minigore. The game uses the players smartphone as a controller, allowing for interesting mini-games and making co-op games with your friends even easier. The objective of the game is to survive to as high a wave as possible, unlocking rooms and even an upgrade shop as you make your way through the level.

TECHNICAL

The main MoleZ game is developed to run on PC or Mac, and the controller app has been developed to run on any Android smartphone. The controller app could be built for iOS devices, however Apple's restrictions on development make it hard to prototype when developing with a Windows PC.

The game and app were both built using Unity development software, as I have the most experience with using this software. The game uses Unity's (now deprecated) networking system, specifically the high-level API, to connect the mobile devices to the main game. The main game runs as a multiplayer server, and the smartphones connect directly as clients. Due to this, the controllers must be on the same network as the main game to allow a successful connection. A direct connection allows messages to be sent between the clients and server at high speed, with almost no latency on a high-quality connection.

RECOMMENDED SPECS

The game has two quality modes to allow it to be played on systems with varying performance.

The low-quality mode can be played on lower end systems, with the recommended specifications to run at 60fps outlined below.

LOW QUALITY MODE

OS: Win 7 or higher / Mac OS Mavericks or higher

CPU: Any CPU made after 2012

GPU: Nvidia GTX 660 or AMD HD 7850

RAM: 4GB

STORAGE: 500 MB

The high-quality mode has fancier lighting effects, as well as post processing effects, however this severely lowers performance on weaker systems, therefore the specifications to run in this mode are higher.

HIGH QUALITY MODE

OS: Win 7 or higher / Mac OS Mavericks or higher

CPU: Any CPU made after 2016

GPU: Nvidia GTX 970 or AMD RX 290X

RAM: 4GB

STORAGE: 500MB

CONTROLS

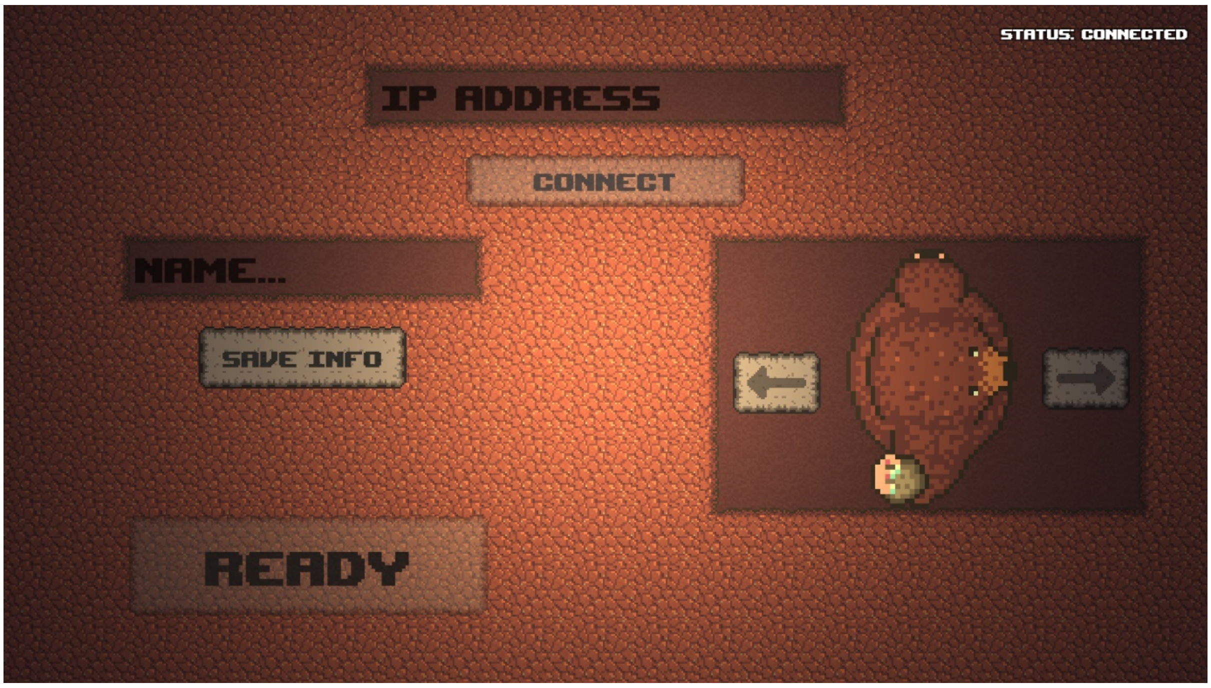
The game uses a player's smartphone as the controller and allows for up to four players to connect at once. Using a smartphone as a controller has benefits and drawbacks, some of which are outlined below.

BENEFITS	**DRAWBACKS**
Most people own a smartphone, so playing co-op games with your friends is even easier.	No tactile feedback when pressing buttons, so can be hard to find specific buttons without looking at the screen.
Content on the controller can be changed to reflect in game scenarios and each player can have their own unique content/control scheme.	Slight latency when inputting button presses.
Game can make use of the phones accelerometer to allow motion controls.	Phone batteries deplete a lot quicker than a standard controller.
A touch screen allows for different types of input other than buttons, such as swipes or pinches.	Different phone screen sizes and operating systems make it hard to develop a single control scheme that works across all devices.

Taking into consideration these pros and cons makes it possible to create an interesting control scheme and allows for innovative, interesting gameplay scenarios. When developing MoleZ, a lot of time was spent making sure that the controls were responsive, intuitive, and fun to use.

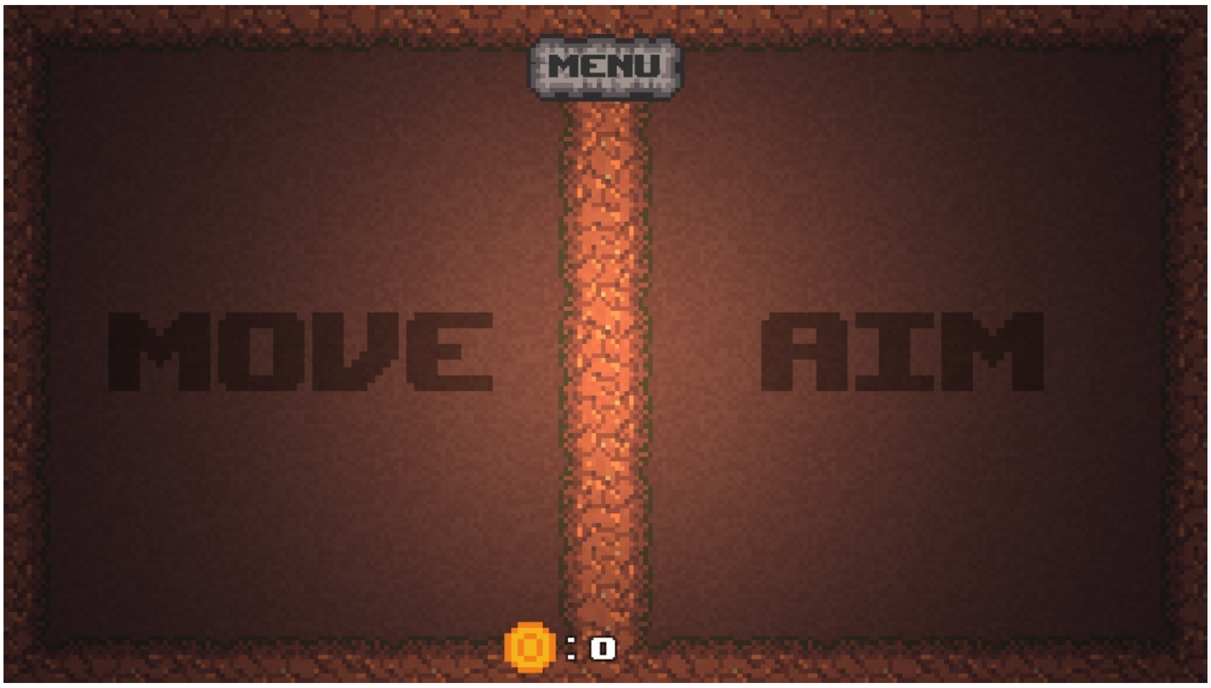
There are multiple controller scenes that are used in MoleZ and below are descriptions of each scene, their purpose and functionality.

PLAYER CONFIG



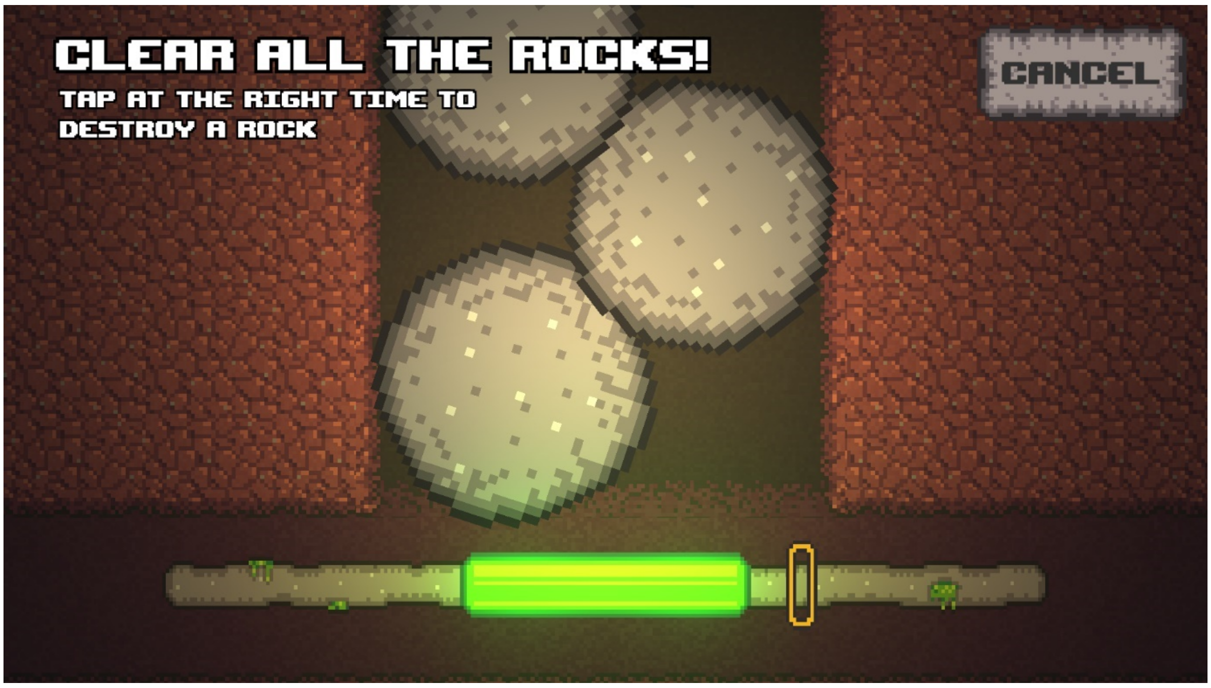
This scene is displayed once the user has connected to the main game server. It takes advantage of the fact that smartphones allow for text input, and allows the player to set their name and choose a skin to play the game with.

MAIN CONTROLLER



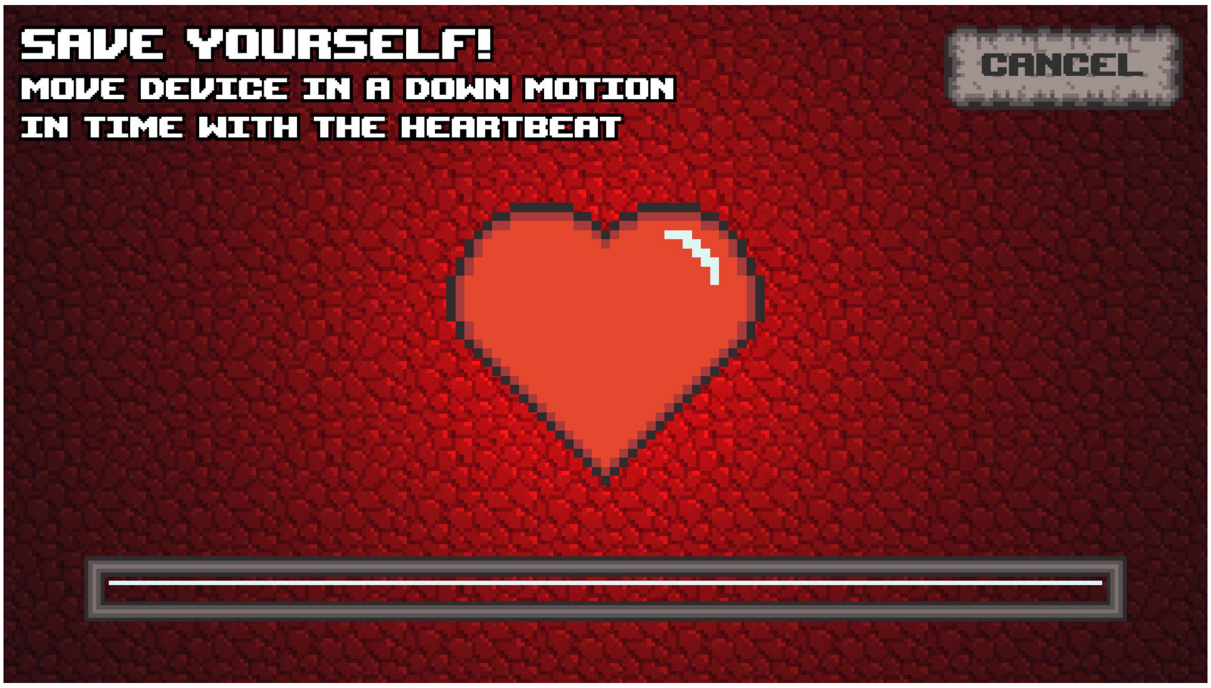
This is the main controller scene and is used to controller the players movement and aiming. When the player touches the left-hand side of the screen, a joystick is created in that location, and then by moving the touch, they can control the characters movement. The aiming works the same as the movement, using the right-hand side instead. The player automatically fires when aiming, to keep the controls simple and easy to use. Having a control scheme like this allows the player to keep their attention on the main game, without having to look down to figure out where to put their thumbs. Also, taking advantage of being able to show different information on each player's phone, the players points are displayed at the bottom centre. At the top is a menu button which allows the player to view a help screen, which gives a slight overview of the game, as well as a quit button, allowing them to leave the game.

DOOR MINIGAME



This scene is opened when a player tries to unlock a door. Taking advantage of the dynamics of the phone screen, a short minigame is displayed. The game is simple, the player has to tap at the right time to destroy a rock, and the game is complete, and door opened, when all rocks are destroyed. The nature of this minigame means that a player has to divert their attention from the main game to their phone, risking taking damage. This has the benefit of creating scenarios in which one player must protect another whilst attempting to open a door and also slows down progression through the map.

REVIVING MINIGAME

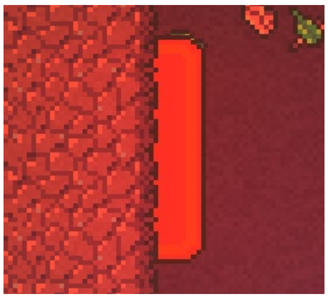


This scene is displayed when a player either, goes down when in a solo game, or begins attempting to revive a player who has gone down in a co-op game. This minigame takes advantage of the phone's accelerometer to make for a fun, motion-controlled game. When in a solo game, and the player is playing alone, the players phone will vibrate in a heartbeat style in a specific rhythm, and the challenge is to move the device, in a 'CPR' like way to match the rhythm of the heartbeat. The heart icon in the centre will also reflect the rhythm of the vibration to add a visual cue to the game. While the player is in time, the bar at the bottom will begin to fill and once full, the minigame will be complete and the player will have revived himself. When in a multiplayer game, the mechanics are essentially the same however have a few small, but key differences. On the player's phone who is attempting to revive, the phone won't vibrate and the heart won't be displayer. Instead the player who is down will have the vibration and heart icon, and the goal is to communicate the rhythm of their heartbeat to the player who is trying to revive them. This forces communication between the players in the real world and adds an element of urgency when trying to revive a downed player.

Overall the controls take advantage of the medium of the smartphone as a controller and work in a way that the player can forget that they are actually using their phone to control the game, to allow increased immersion.

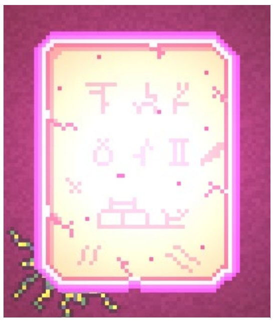
MECHANICS

There are many mechanics in MoleZ that add to the complexity of the game. As the player is always shooting whilst aiming, there is a need to have infinite ammo. The map is split into multiple sections separate by doors. The game uses a wave mechanic, with each wave increasing in difficulty by increasing the number of zombies, and the strength of the zombies. Between each wave, there is a short interval to allow players to unlock doors and visit the shop. The wave number, number of zombies, and wave interval are all displayed on screen using text to inform players of these different values.

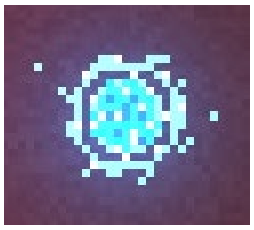


Once the game has started, there is a button on the wall that all players must press to start the first wave. This is to allow players to read the help screen, which displays when first launching the controller app, and familiarise themselves with the controls before having to deal with zombies. This was added as a result of playtesting, in which new players were overwhelmed when first entering the game, and didn't have enough time to fully understand the controls before having to defend themselves. Once all players have pressed the button, it disappears and the first wave begins shortly after.

Occasionally, a heavy zombie will spawn. These are slightly larger than the standard zombie, and have increased health.



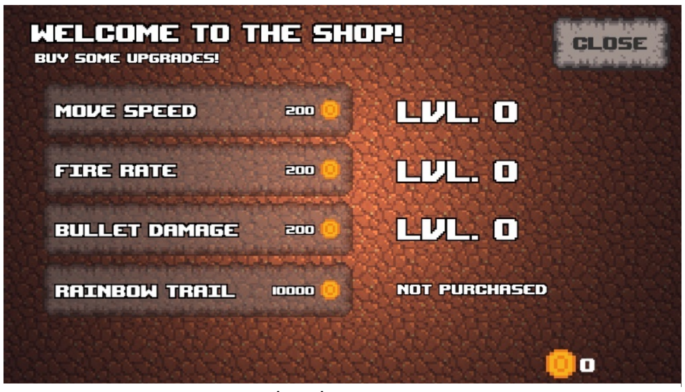
Permanent Upgrade



Temporary Upgrade

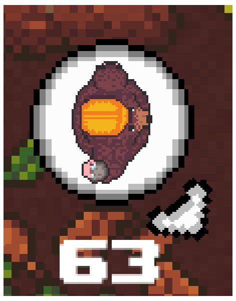
There are different types of upgrades that can be found throughout the game. One is permanent, which either increases the players movement speed or fire rate. These upgrades are found in certain rooms and permanently buff the player's stats for the current game. The next is a temporary upgrade, which can be found occasionally dropped when killing a zombie. These last five seconds and apply one of two temporary buffs to the player. One is a rapid-fire upgrade, which massively increases the players fire rate, and the other is a double speed upgrade, which allows the player to quickly run past zombies before they even have a chance to damage the player. There is also a health pickup, which also has a chance to spawn when a zombie dies. These are pretty self-explanatory, and simply add a heart to the players health if they are missing one or more. The last upgrades are found in the shop, which is unlocked once the player has opened every door in the map.

The shop contains four different upgrades, three of which have incremental levels. The shop allows players to get further in the game once they have unlocked the whole map, and adds a reason to keep playing. The three incremental upgrades are a move speed buff, a fire rate buff, and a bullet damage buff. The player uses points gained from damaging and killing zombies to buy these upgrades, and they increase in price exponentially the higher level each upgrade is. There is no cap to the maximum level of each upgrade, however as the cost increases it becomes increasing harder to save up points to buy each upgrade, preventing players from becoming overpowered. In the later rounds, these upgrades are crucial as the zombies increase in strength and numbers as the game progresses. Finally, the last upgrade in the shop is a rainbow trail upgrade. This upgrade has a high fixed price and can only be purchased once. This upgrade is purely cosmetic, and simply adds a rainbow particle effect that trails behind the player as they move. This upgrade adds a fun goal for more experienced players to achieve.



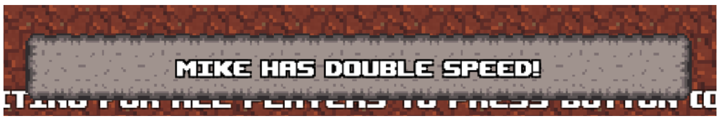
The Shop Menu

When in a multiplayer game, the camera dynamically zooms and pans to keep all the players on the screen. The camera movement is smoothed to not be annoying, however this has the draw back of occasionally falling behind and letting a player go out of view, especially when the player has the double speed temporary upgrade. While the camera catches up, a small indicator is placed on the edge of the screen to give a slight indication of where the player is. The camera also dynamically changes which targets it follows, for example, when a player goes down, they are no longer tracked by the camera. Once again, in this scenario, a small indicator is placed on the edge of the screen, informing any players left alive where the downed player is and also how long is left to revive them before they die.



Off-Screen Indicator

There is also a popup notification system in place to inform players when different events happen in game. These appear at the bottom, and simply contain text dependent on the event. These happen when an upgrade is picked up, when the shop is unlocked, and when a player goes down.



Pop-up Notification

The zombie pathfinding is achieved using Navigation2D (https://assetstore.unity.com/packages/tools/ai/navigation2d-pathfinding-for-2d-games-

35803). This premium asset functions as a wrapper for Unity's in built navigation system to allow it to work in 2D games. Currently, Unity's navigation system is limited to only 3D games, however with many axis transformations it is possible to use it in 2D games. This asset works well to wrap almost all in built navigation classes, such as NavMeshAgent, and makes them work almost flawlessly.

CLASSES

This is a list of all the classes in the main server game with a short description of their usage.

Arm.cs

Attached to the players arm, this class contains a function that runs when the throwing animation is complete.

Bullet.cs

Attached to each bullet, this class contains information relating to each bullet, such as its damage, speed and the player who threw it.

CharController.cs

This class controls is attached to each player and controls their movement. Also controls initiating interactions.

DontDestroy.cs

This class is attached to the background music object to ensure that the music doesn't stop playing between scenes.

Door.cs

Attached to each door, this class contains a reference to the room it is attached to, and handles unlocking of the door.

Fader.cs

Contains functions that are run once the scene fader has finished its animation.

GameManager.cs

This class handles almost all network messages received while in game, and also the opening of the shop. Also handles sending messages to clients phone, such as beginning minigames or opening the shop.

GameOver.cs

This class runs when the game over scene is loaded. Resets all static variables and shuts down the network server.

HealthPickup.cs

Attached to the health pickup, detects when a player enters the collider and adds health.

Interactable.cs

Attached to interactable objects, has a type drop down in the inspector to decide what is done when the object is interacted with.

NetworkServerUI.cs

This class handles connecting players initially, and setting up player objects with the connected players information. Displays the connected players information. Starts the game once all players are ready.

OffScreenIndicators.cs

Handles displaying and positioning the off-screen indicators which show when a player isn't in the camera view.

PermUpgrade.cs

This class is attached to the permanent upgrade pick up. Handles collision and applying effects.

Player.cs

An empty Player object, contains name, colour, connectionID, ready status, a reference to the CharController.cs, a reference to the players GameObject and whether the player is dead.

PlayerHealth.cs

Manages all the player health functions. Has functions for when the player goes down, when the player is revived, and when they die. Also handles displaying the health icons above each player.

PlayerManager.cs

Holds a list with all connected players. Also contains a list of possible player sprites.

PopUp.cs

Attached to each pop up, contains a function to destroy it when animation is complete.

PopUpManager.cs

Controls creation of pop ups. Has a static method to allow easy creation of popups from anywhere.

QualityManager.cs

Controls the quality level of the game. Disables fancy dynamic lights and adjusts ambient light settings.

Room.cs

Attached to each room. Contains lists of everything in the room such as lights, doors and spawn locations. Has functions to show all these when a room is opened.

Shop.cs

Handles getting the player’s upgrade values and sending them to the players controller when a player opens the shop.

ShopValues.cs

Stores all the players upgrade values and contains a function to update them once a player has finished in the shop.

StartBtnScripts.cs

Handles all the buttons on the first scene of the game, such as the music toggle, the quality toggle and the quit button.

StartButton.cs

Attached to the button that is in the first room, that all players must press to start the game. Waits until all players have pressed it then starts the first wave.

TempUpgrade.cs

Attached to the temporary upgrade pickup to handle player collision and applying the upgrade.

TorchFlicker.cs

Small class to make the point lights attached to torches flicker by changing their intensity.

TrackTargets.cs

Attached to the camera in the main scene, this class controls the panning and zooming off the camera to keep all the players in view. Contains functions to update targets.

Weapon.cs

Handles the players weapon functionality, mainly playing the throw animation and instatiating bullets.

ZombieAttack.cs

Handles the zombie attacking functions.

ZombieHealth.cs

Controls the zombie health and dropping of temporary upgrades / health drops. Also controls the animation.

ZombieManager.cs

This class controls all of the zombies, such as the wave system and spawning. Calculates the amount of zombies per wave as well as the starting health of each zombie.

ZombieMove.cs

Controls the zombie movement using NavMeshAgent2D. Also controls rotation to make the zombies face the direction they are moving.

This is a list of all the classes in the client controller app with a short description of their usage.

ControllerButtons.cs

Handles functionality of the menu buttons in the main controller scene.

DeathScreen.cs

Sets the kill count text and resets the static variables before resetting the app.

DoorButtons.cs

Handles the cancel button functionality in the door unlocking minigame.

DoorMinigame.cs

Controls all mechanics of the door minigame, such as spawning rocks, setting difficulty and sending a message to the server when the game is complete.

DownedButtons.cs

Handles the cancel button functionality in the downed minigame.

Fader.cs

Contains functions that run when the fader completes its animation

Joystick.cs

Handles the touch input on the main controller scene and also creating the joysticks.

NetworkClientUI.cs

Registers all the message handlers for the network client to receive messages. Handles sending messages to the server. All connections to the server go through this class. Also controls the UI on the config screen where the player inputs name and skin.

PlayerConfig.cs

Saves the players config values locally to static variables.

PlayerConfigBtns.cs

Controls the functionality of the configuration buttons.

Points.cs

Contains a static variable with the players current points.

ReviveMinigame.cs

Handles all mechanics for the reviving minigame, such as vibration, and detecting gestures.

ReviveVibration.cs

Used in co-op games, handles the vibration on the downed players phone.

Shop.cs

Handles all functionality of the shop menu, such as calculating prices and sending new levels back to the server.

StartScreenBtn.cs

Contains the function that runs when the connect button is pressed.

TitleScripts.cs

Controls the logo fade and text on the title screen.

TorchFlicker.cs

Small class to make the point lights attached to torches flicker by changing their intensity.

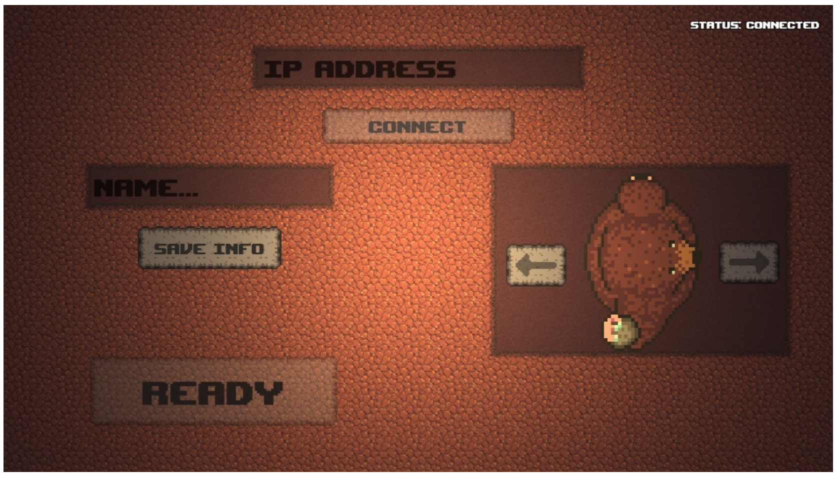
Vibration.cs

Class from https://gist.github.com/aVolpe/707c8cf46b1bb8dfb363

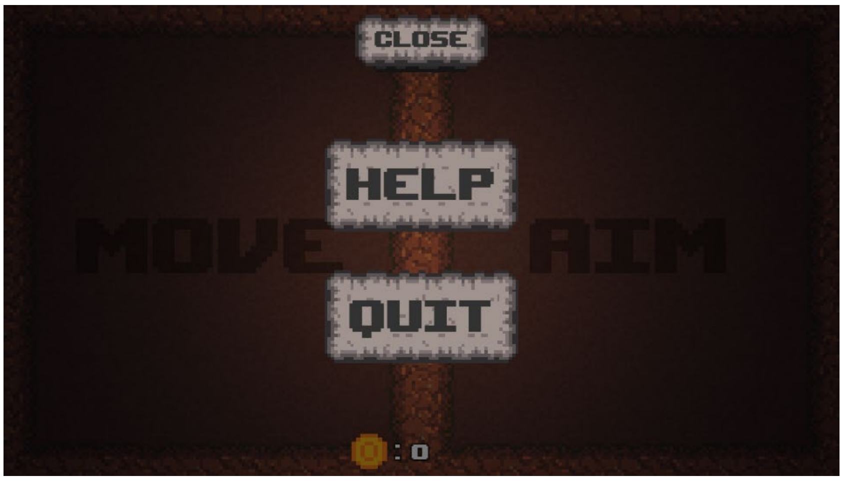
Allows extended vibration functionality such as length and vibration patterns.

UI DESIGN

The UI for MoleZ is designed in a way to make it fit within the game world. As the main interaction with the game is through a mobile device, the buttons and text needed to be bold and clear.



The title page on the app is displayed first, and the player can tap anywhere to switch to the connection screen. Once on the connection screen, there is an input field, and a connect button. The art for these is made from the same art that is used in the level design, to make all aspects feel as part of one larger whole. Once connected, the player configuration window appears and players can input their name, and chose a skin for their character. There is a save info button that allows a player to save the information they have decided on, and to make sure the player chooses a name, the button to ready up is only enabled once the save info button has been pressed.



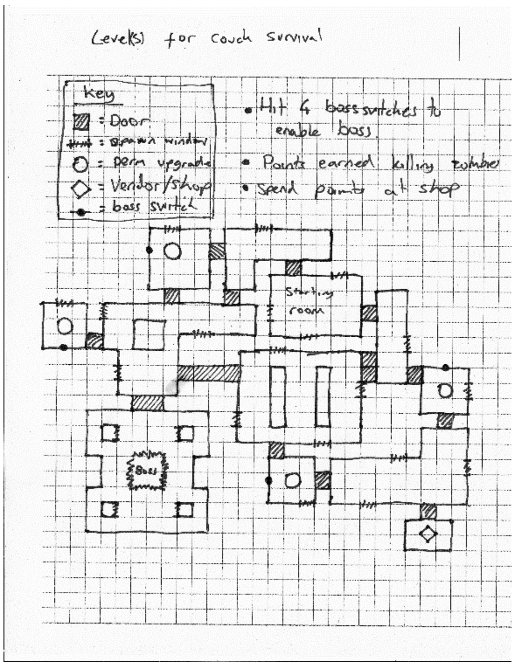
Once in the controller scene, on first launch of the app a help screen is displayed. This contains text to inform the player how to control the game. The controller scene has a menu button at the top centre, out of the way of the input areas to not interrupt gameplay, however large enough to make it easy to press when needed. On press, this button opens a simple menu with two large buttons, one to reopen the help menu, and one to let the player quit from the current game.

The UI uses a consistent styling throughout, and all buttons are clearly labelled.

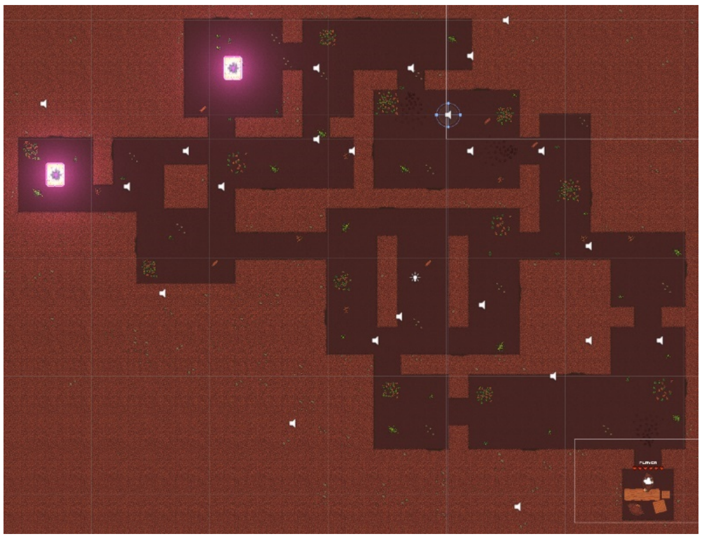
LEVEL DESIGN

The level design of MoleZ uses a system of rooms and corridors. Each room is a unique shape to give variety to the level design. The idea with the level design in MoleZ is to give a sense of exploration and choice when the player moves throughout the level. By making no two rooms the same, the player will constantly be surprised when entering a new room. The level creation uses Unity's Tilemap system, so the layout is based on a grid. This tile system allowed for easy prototyping of level designs and room shapes, and Unity even features a tilemap collider that makes placing wall colliders easier.

The initial sketches had plans for a boss room, however time constraints meant this couldn't be implemented.



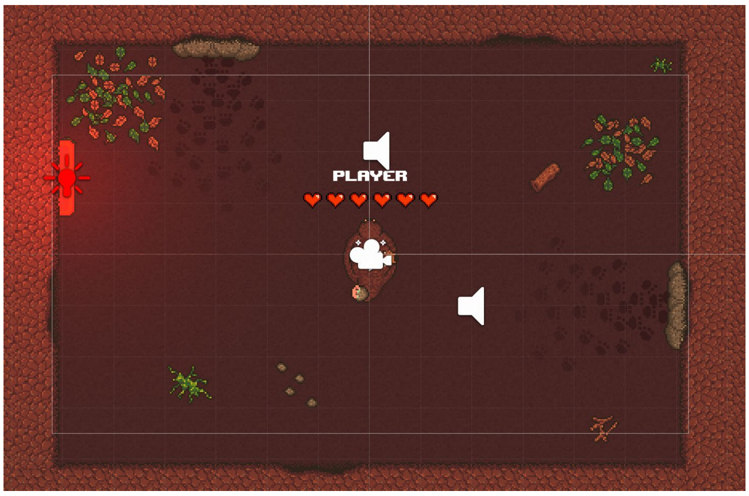
Initial Sketch for Level Design



The Whole Unlocked Level

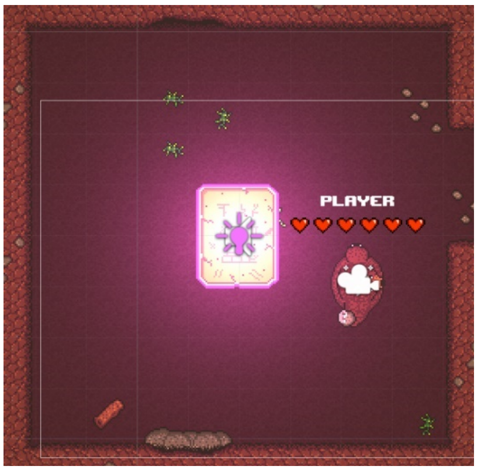
FIRST ROOM

The first room the players start in is a simple rectangular shape. There is quite a lot of free space in this room to allow a player to get used to moving around and shooting zombies. There are two doors in the first room to give the player a choice of which one to take, allowing



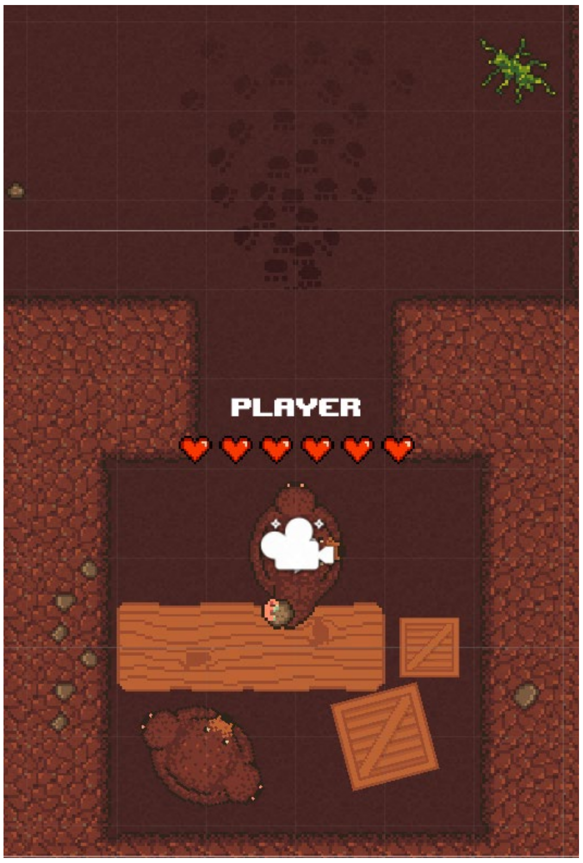
the player to feel a sense of freedom in exploration. To indicate where the doors are to new players, this first room has foot prints on the floor to indicate where a player should go.

PERM UPGRADE ROOMS



Around the map there are four rooms that contain permanent upgrades. These all have a square design and are usually require opening one more door to get to the next main room. These rooms give the player a reason to explore the map, and act as a reward for opening the room. There are four in the map to allow each player in a full game to collect one, if all the players are fair.

THE SHOP



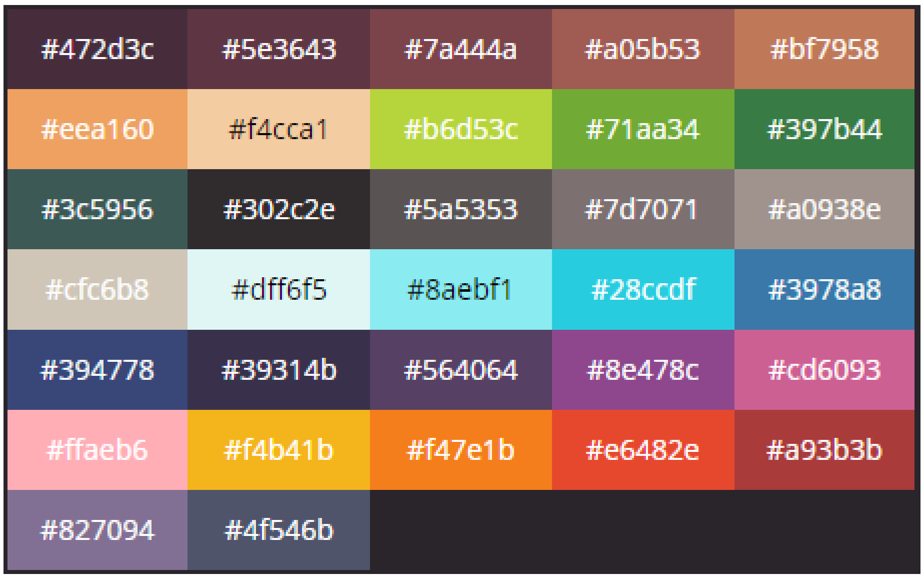
The shop is the only hidden room in the level. The entry to this room doesn't have a conventional door like all other rooms, and can only be opened by unlocking all twelve doors in the map. The only indication this secret room exists are the footprints on the floor that seem to lead into a wall, and the fact each player has a coin count on their phone controller. When this room is opened, a small popup appears to inform the players. The room features a mole behind a counter with some boxes. The shop gives players who have unlocked the whole map a reason to keep playing, and also increases replayability, as different combinations of upgrades can be tested out. It also adds a way for a player to make later rounds slightly easier, and even purchase a useless cosmetic upgrade. It can also give players who know about its existence a reason to explore the whole map, and unlock all the doors.

ART

All of MoleZ art assets are made using Photoshop CC in a pixel art style. The pixel art style was chosen as it has a clear stylistic aesthetic of an arcade style video game.

To keep all art assets looks as part of one larger whole, MoleZ uses a restricted colour palette. Almost all colours in the game are from this palette apart from a few select exceptions.

The palette used is the Zughy 32 palette found at https://lospec.com/palette-list/zughy-32. This palette has a variety of shades for the main colours needed for the game, such as earthy colours.

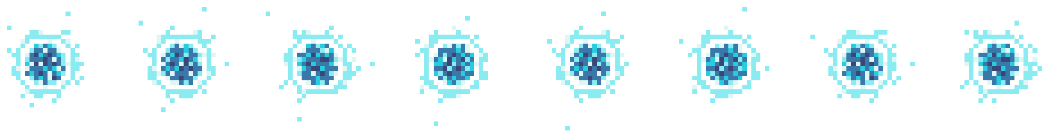


#472d3c	#5e3643	#7a444a	#a05b53	#bf7958
#eea160	#f4cca1	#b6d53c	#71aa34	#397b44
#3c5956	#302c2e	#5a5353	#7d7071	#a0938e
#cfc6b8	#dff6f5	#8aebf1	#28ccdf	#3978a8
#394778	#39314b	#564064	#8e478c	#cd6093
#ffaeb6	#f4b41b	#f47e1b	#e6482e	#a93b3b
#827094	#4f546b			

Zughy 32 Palette

The level assets are all part of a single tile sheet, to be used in Unity's Tilemap system. Each tile is 32x32 pixels.

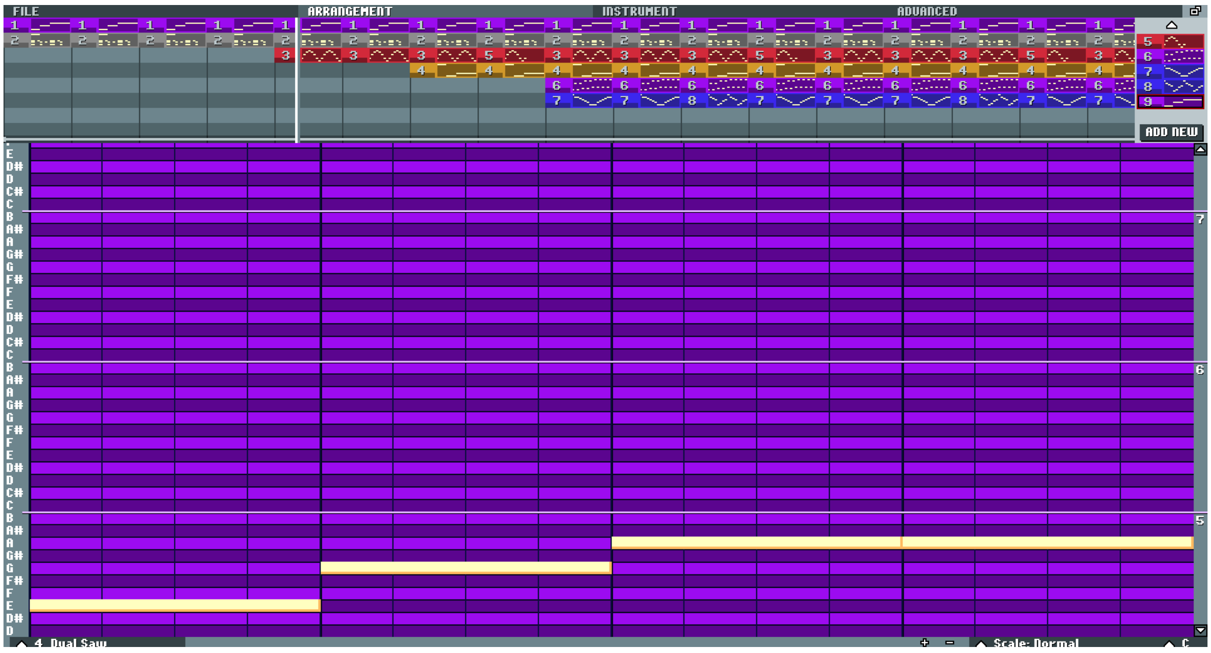
All animation in MoleZ are limited, with the largest using eight individual sprites. The animations were made using Photoshop's frame animation tool, and then exported to a single sprite sheet using the Photoshop script "Spritesheet Generator.jsx" found at https://github.com/bogdanrybak/spritesheet-generator. This script takes the individual frames of the animation and creates a single sprite sheet using the dimensions supplied. This can then be imported into Unity and used as an animation.



Temporary Upgrade Sprite Sheet

SOUND

MoleZ has one main theme song. The theme song is writted using a blues scale in the key of E and features many layers. The music was created using “Bosca Ceoil”, a program that can be used to created video game music using retro chip tune sounds.



The program works by making short four bar segments, and then arranging them to make up the full song. Each segment can be given its own instrument which can be tweaked until it sounds right. The program also allows setting a scale and key for the composer section, to make music creation easier. Once done, it allows exporting to a .wav file which can then be used directly within Unity.

Bosca Ceoil is a free program and can be found at https://boscaceoil.net/.

Bosca Ceoil was used as it allowed for easy straight forward music creation, in an ‘arcadey’ style which fit with the game’s aesthetic and gameplay.

All sound effects used in game are Creative Commons licensed sounds attained from https://freesound.org/. Some sound effects were slightly tweaked in Adobe Audition to make them fit better with their use case.

A full list of sound effects and their individual sources.

NAME	LINK
throw1	https://freesound.org/people/qubodup/sounds/60013/
throw2	https://freesound.org/people/qubodup/sounds/60009/
bullet hit	https://freesound.org/people/Aurelon/sounds/422633/
Player hurt1 to 4	https://freesound.org/people/Artmasterrich/sounds/345438/

	https://freesound.org/people/Artmasterrich/sounds/345439/ https://freesound.org/people/Artmasterrich/sounds/345436/ https://freesound.org/people/Artmasterrich/sounds/345433/
Zombie groan	https://freesound.org/people/JarAxe/packs/11408/
Wave beginning	https://freesound.org/people/JarAxe/sounds/92026/
Zombie hurt1 & 2	https://freesound.org/people/Artmasterrich/sounds/345439/ https://freesound.org/people/Artmasterrich/sounds/345436/
Door opening	https://freesound.org/people/adamgryu/sounds/336023/
Temporary upgrade	https://freesound.org/people/TreasureSounds/sounds/332629/
Permanent upgrade	https://freesound.org/people/plasterbrain/sounds/464902/
Health pickup	https://freesound.org/people/ScratchnSniff/sounds/242113/

All sounds in the game apart from the zombie groan, use a 2D play mode so they play equally loud wherever the player is. The zombie groan uses a 3D spatial mode so they will player proportional out of the left and right speaker depending on their relation to the camera's location. This has the effect of letting players know which direction a zombie is coming from.

ASSETS

MAIN GAME ASSETS

TYPE	FILE NAME	FIRST APPEARANCE (SCENE NAME)	DESCRIPTION OF USAGE
Font	upheavtt	Start	Font
Material	Part	Level	Particle material
Material	SpriteDiffuse	Start	Sprite material to interact with lights
Tilemap Prefab	door	Level	Door
Tilemap Prefab	zombie_spawn	Level	Zombie spawn location
Prefab	Bullet	Level	Bullet
Prefab	Character	Level	Player controlled character
Prefab	Fader	Start	Used to fade between scenes
Prefab	health	Level	Used to show players health
Prefab	HealthPickup	Level	Dropped by zombies, adds health to player.
Prefab	OffscreenIndicator	Level	Shows direction to an offscreen player
Prefab	torch	Start	A flaming torch with light attached
Prefab	UpgradePerm	Level	Permanent Upgrade
Prefab	UpgradeTemp	Level	Temporary Upgrade
Prefab	Zombie	Level	A zombie
Prefab	PopUpCanvas	Level	A Blank popup notification
Sprite	mole	Start	Image of default mole to show user skin choice

Sprite	mole-bow	Start	Image of mole with bow to show user skin choice
Sprite	mole-hat	Start	Image of mole with hat to show user skin choice
Sprite	mole-glasses	Start	Image of mole with glasses to show user skin choice
Scene	Start	Start	Start Scene
Scene	Level	Level	Level Scene
Scene	GameOver	GameOver	Game Over Scene
Music	song1	Start	Background Music
SFX	bullet-hit	Level	Sound when bullet breaks
SFX	door-opening	Level	Sound of door opening
SFX	health-pickup	Level	Sound when health is picked up
SFX	perm-pickup	Level	Sound when permanent upgrade is picked up
SFX	temp-pickup	Level	Sound when temporary upgrade is picked up
SFX	wave-start	Level	Sound played when wave is beginning
SFX	hurt1 to hurt4	Level	Sound played when a player is hurt
SFX	throw1/throw2	Level	Sound played when a player throws
SFX	hurt1/hurt2	Level	Sound played when a zombie is damaged

SFX	groan1 to groan 12	Level	Zombie groan sounds
Sprite Sheet	badger-die	Level	Badger dying animation
Sprite Sheet	badger-walk	Level	Badger walk animation
Tile Sheet	tiles	Level	Tile sheet for map creation
Sprite	button	Level	Button to start first wave
Sprite	rune	Level	Permanent upgrade sprite
Sprite Sheet	temp-upgrade	Level	Temporary upgrade animation
Sprite Sheet	torch	Start	Torch animation
Sprite Sheet	mole	Level	Default mole animation
Sprite Sheet	mole-arm	Level	Mole arm animation
Sprite Sheet	mole-bow	Level	Mole with bow animation
Sprite Sheet	mole-glasses	Level	Mole with glasses animation
Sprite Sheet	mole-hat	Level	Mole with hat animation
Sprite	health	Level	Health icon
Sprite	stone	Level	Bullet sprite
Sprite	arrow	Level	Off screen indicator arrow
Sprite	frame	Level	Off screen indicator frame
Sprite	bench	Level	Bench in shop
Sprite	crate	Level	Crate in shop
Sprite	leaves1 to leaves 5	Level	Leaves on floor prop
Sprite	vendor	Level	Vendor in shop
Sprite	background	Start	Background of connection screen
Sprite	BannerLogo	Launcher	Banner displayed on

			launcher before game starts
Sprite	button	Start	Button sprite
Icon	icon	Game Icon	Icon for game
Sprite	Logo	Start	Logo on connection screen
Sprite	player-container	Start	Container the connected players are displayed in
Sprite	ready	Start	Icon showing if player is ready
Sprite	unready	Start	Icon showing if player is not ready
Sprite	tfgameslogo	Splash Screen	Developer logo

APP ASSETS

TYPE	**FILE NAME**	**FIRST APPEARANCE (SCENE NAME)**	**DESCRIPTION OF USAGE**
Font	upheavtt	Title	Font
Material	SpriteDiffuse	Title	Sprite material to allow light to affect
Prefab	Fader	Title	Prefab to fade between scenes
Prefab	rock	DoorMinigame	Rock spawned in the door minigame
Sprite	mole	Start	default mole to display the players skin choice
Sprite	mole-bow	Start	mole with bow to display the players skin choice
Sprite	mole-glasses	Start	mole with glasses to display the players skin choice
Sprite	mole-hat	Start	mole with hat to display the players skin choice
Scene	Controller	Controller	Main controller scene
Scene	Dead	Dead	Showed when player dies

Scene	DoorMinigame	DoorMinigame	The door minigame
Scene	DownedMinigame	DownedMinigame	The downed/reviving minigame
Scene	DownedMulti	DownedMulti	Scene displayed when player goes down in co-op game
Scene	Shop	Shop	Shop menu
Scene	Start	Start	The scene where player connects to server and choses name/skin
Scene	Title	Title	The title scene where logo is displayed
Sprite	aim-loc	Controller	Part of aiming joystick to show direction
Sprite	coin	Controller	Small coin icon shown next to points amount
Sprite	move-loc	Controller	Part of movement joystick to show direction
Sprite	stick-origin	Controller	The base of the joystick
Sprite	bar	DoorMinigame	The bar at bottom of screen in door minigame
Sprite	door	DoorMinigame	The area in which rocks spawn
Sprite	rock	DoorMinigame	Sprite for the rock
Sprite	slider	DoorMinigame	Slider that moves back and forth along the bar
Sprite	bar	DownedMinigame	The empty bar sprite
Sprite	fill	DownedMinigame	The sprite that fills the empty bar as player is in time
Sprite	heart	DownedMinigame DownedMulti	Heart icon that beats in time with the rhythm

Sprite	arrow	Start	Icon for the prev/next buttons on the character selection
Sprite	background	Title	The tiled background
Sprite	button	Start	The button art
Sprite	container-thin	Start	The background of the input fields
Sprite	Icon	Game Icon	Icon
Sprite	Logo	Title	Logo displayed on title
Sprite	tfgameslogo	Splash Screen	Developer logo

TARGET AUDIENCE

The target audience for MoleZ is all game players. The easy to use controls cater to players who have never even used a standard game controller before, allowing them to quickly understand how to play in an intuitive way. The game also begins off quite easy to allow unexperienced players to feel a sense of accomplishment. The game also caters to more experienced players by increasing in difficulty towards the higher rounds and the opening of the upgrade shop giving hardcore gamers something to do to improve their high score.

The game is meant to be played in co-op mode, allowing groups of friends to play together in the same room. "Couch co-op" is almost disappearing in todays gaming market, in favour of online multiplayer, however this doesn't have the same feel as being able to sit down in a room with your friends and play a game.

In addition to the co-op mode, the game also supports a solo play, with slightly different mechanics to allow players to experience and play the game alone. The slight changes in mechanics mean the game is equally as challenging when playing alone.

LICENSE

MoleZ used multiple softwares during its creation. Licenses are below.

Engine

Unity 2018.1.6.f1 Personal License

Development engine

Available at https://unity3d.com/get-unity/download/archive

Art

Adobe Photoshop CC

Art creation software

Available at https://www.adobe.com/uk/products/photoshop.html

Audio

Bosca Ceoil

Video game music creation software

Available at https://boscaceoil.net/

Adobe Audition CC

Sound editing software

Available at https://www.adobe.com/uk/products/audition.html

CRITICAL REFLECTION

Since beginning this project, I have learnt many things regarding developing games in Unity. Beforehand, I only had limited experience using the software, and this project allowed me to explore many different directions that I had never looked into before. I feel that overall, the end game works in the way I had imagined at the start.

The first major step for me was to understand how networking works in Unity. This feature of Unity is now deprecated, and their replacement does not work in the same way, and wouldn't have allowed me to make the game in the way I did. Because of this, I had to use an older version of Unity, 2018.1.6f1 to be exact, which allowed setting up of a network server and clients. MoleZ runs in the same way a multiplayer game does, with a server and client, however there are a few key differences. Instead of the server synchronising data across all the clients, it instead acts as the main game. The controller app then acts as the client and using network messages, it's possible to get the two to communicate with each other almost instantaneously. Thanks to Unity's networking system, I was able to get the communication between the game and app working a lot quicker than expected, and allowed me to spend more time on polishing and refining the game. I was originally worried about latency too, as the controller inputs from the app would have to be sent to the main server. After extensive testing it seems that latency is so miniscule that it isn't an issue. I expect this is due to the direct connection between the game server and app on a local network. Overall the aspect of using a phone as a controller for a game works better than I had imagined, and allowed me to work on creating unique minigames.

When planning on how to create the game, I initially planned to use Unity's navigation system to control the zombie's pathfinding. When it came to implement it, I was disheartened to discover that the navigation system only natively supports 3D games. This was a major setback for me, as I had built the whole game planning to use it. I researched many different ways around it, such as building the game in 3D and using a locked camera perspective, or implementing my own pathfinding algorithm such as A*. Both of these implementations would have taken a considerable amount of time to implement, so in the end I decided to use a premium asset from the Unity Asset store. The asset I chose, named Navigation2D, acts as a wrapper for the built in navigation system, by translating all of the navigation agents and navigation mesh over to a 3D plane to calculate navigation, then translating back to position the sprites in a 2D environment. This asset helped immensely, and save me a large amount of time. While not flawless, the asset worked well enough for the use case I needed it for, and I even found and fixed a small bug in the asset code when it came to displaying gizmos in the scene view, which I communicated back to the asset author with the fix now being present in the latest version.

When it came to creating the art assets for the game, I was worried as I don't consider myself an artist at all, and have almost no experience creating 2D assets for a game especially animations. I decided to settle on a pixel art style as I felt it fits the 'arcadey' style of the game, as well as personally finding it easier to create. I chose to use a limited palette to help refine the style of the game as well. I feel the tiling sprites used to create the level ended up looking really good, however I would have liked the animations to be a bit more refined. Overall, while basic, the art in the game follows a clear style and by using the same assets in the controller app and main game, it helps them to feel as part of one product.

Creating sound was also a worry for me, while I was the sound artist on my last project, I still have limited experience so I researched to find an appropriate software that would allow me to easily create music for the game. I chose Bosca Ceoil as it simplified creating music down into four bar segments, which could then be arranged however I wanted. The music created also has a video game sound to it, which worked perfectly in the situation I wanted it. I chose to use royalty free sound effects from freesound.org to use as sound assets in the game, as it allowed me to quickly test out different use cases, and they were far better than anything I could have produced myself in any reasonable amount of time.

I wanted to overcome some of the main obstacles presented when using a phone as a controller, for example, the lack of tactile feedback. The final controller works in a way that only requires direct attention from the player when it is intended, such as in minigames, and allows the player the focus on the main game while controlling their character. This has the effect of removing the difficulty from using a phone as controller and, when testing, many players said they had completely forgotten that they were actually using their phone as a controller and were completely focused on the main game.

If I had had more time, I would've liked to add a few more features to the game. One idea was a boss room, that would require the players to hit four hidden switches around the map to open it. Once the players enter the room, the door locks behind them and normal waves are interrupted as a boss wave begins. The boss would've been a stronger slower version of the normal enemy who the players would have to kill before leaving the room. I chose not to add this as it wasn't essential to make the main game feel complete, and would've added a considerable amount of development time. Another feature I would've liked to add would have been more zombie variety. Currently there are two varieties of the zombie, one slightly heavier. I had plans for a faster zombie that rushes the player, and also one that could explode doing splash damage. I ran out of time when it came to adding these, so decided to instead focus my time on other areas. One final addition I would've added if I had more time would be more upgrades in the shop. The shop as it currently is, has four upgrades. Three player buffs and one cosmetic upgrade. I would've liked more cosmetic upgrades, and maybe some fire modifiers such as burst shot or critical hit chance. If I were continuing to develop MoleZ I would definitely add these, to increase replayability and to give the players even more of an objective after unlocking the shop.

One criticism I want to note about Unity are its bugs. Often, I would encounter a bug whilst developing and then spend a considerable amount of time trying to fix it. I would eventually give up, restart Unity and then magically the bug would be fixed. Also, occasionally I'd restart and there would be new bugs present that weren't there before restarting. This aspect of the development software makes it hard to reliably bug fix and because of this, I am looking to use a different engine when it comes to developing my next game.

Overall, I am pleased with the final MoleZ game. The game has no noticeable or immediate bugs and the aspect of using a phone as a controller makes for a fun and interesting game, and the minigames to complete objectives in the main game make for unique experiences. I've enjoyed developing and playtesting with friends, and whenever the game is being played, all the players get into it and enjoy themselves as well. I feel the game has ended up better than I had initially expected and I have enjoyed every aspect of the development process.