

Tallowmere 2

Big Game, Little Systems

A look at creating a
2D RPG platformer using Unity







with **Chris McFarland**

Independent Game Developer

The original *Tallowmere*

Made with  Unity:

- Violent 2D dungeon platformer
- Random rooms, random loot
- Single-player and 4-player couch co-op
-  Steam,  iOS,  Android,  Wii U
- Over 15,000 copies sold since March 2015
- Fun "how far you can make it" gameplay

Criticisms:

- Hardly any animation
- Not enough weapon and enemy variety
- Lacking story and progression
- Repetitive room theme



Enter *Tallowmere 2*

What does *Tallowmere 2* mean to me? **Bigger scope!**

- Better animation and art (and more of it)
- Stories, progression, unlockables
- Overworld map
- Room themes
- Reusable `Creature` class
- More weapons and enemy mechanics
- Room and enemy modifiers
- Local couch co-op (again)
- Multiple platforms (again)
- Network play
- Mod support



Creatures

What makes a Creature?

- Default structure is humanoid
- Lots of Transforms
- Ping-pong stretching breathes life
- Animation controlled by Timers and states/enums
- Hard-coded animations (I don't use Unity's Animator)
- Gibs upon death



Sprite Z-ordering:

- Skin, Garments, and Items use same sprite layer
- Offset each Transform's local Z position for layering

States & Timers

Using enums as states:

- Most classes in *Tallowmere 2* use an `enum` of some sort
- Traverse enums using `switch` statements
- For animations, desired `position`, `scale`, `rotation`, and `duration` are stored for each state

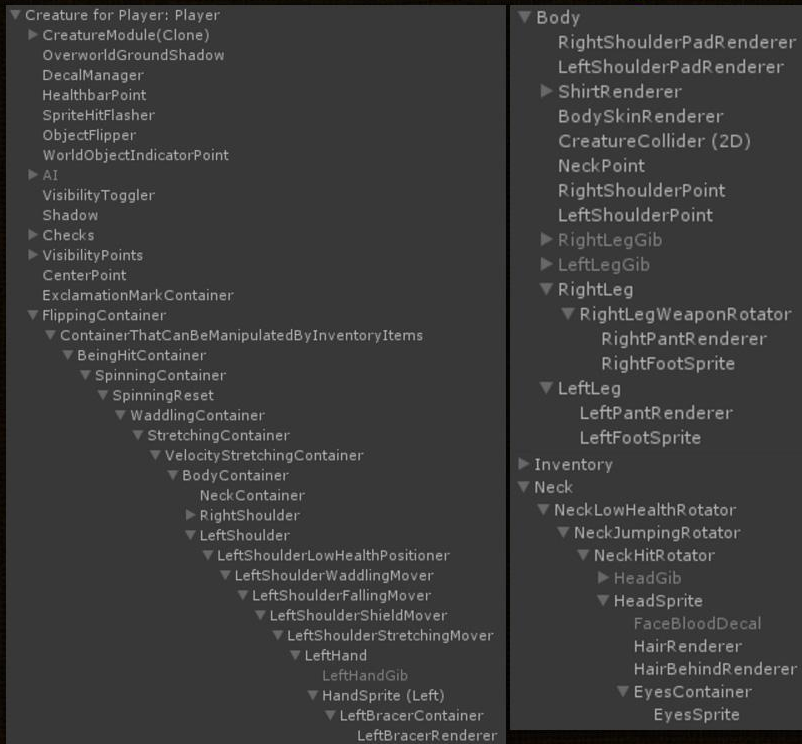
The convenience of timers:

- Timers store the `start time` and `end time` of each animation phase
- Lerp between each position/scale/rotation to achieve the animation (`Vector3.Lerp`, `Quaternion.Lerp`)
- Combine with `Mathf.SmoothStep` to ease the animations nicely
- If timer is done, change state and set timer's new duration



Creature Transforms

What does a Creature's hierarchy look like?



Lots of Transforms:

- Structured like a skeleton (kind of), ready for Sprites to be applied
- Certain methods manipulate certain "containers"



Result:

- Cascading animation from multiple manipulators
- Simplistic yet effective

Creature Animation

Animation = Processing various states and manipulating the Transforms.

MovementState:

- Null, MovingWest, MovingEast
- Controls hands and legs with a further WaddlingState

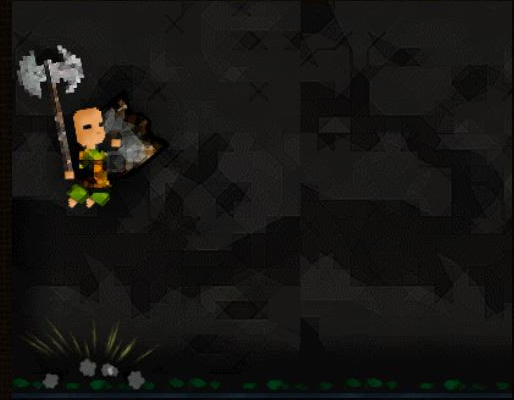
MeleeWeaponState:

- Idle, Raising, SwingingDown, Down, ReturningToIdle
- Controls hand positions; rotates the hind leg and body

ShieldState:

- Lowered, Raising, Raised, Lowering
- Positions the hand; rotates the Shield

Also: Jumping, GettingHit, Blinking



The Player Class

A Player is a logic-only class.

The Player updates their InputActions when they press a button:

- Move west, east
- Jump
- Use weapon
- Toggle shield
- Menu navigation

Keep your Creature separate from your Player:

- A `Creature creature;` variable lets the Player control any Creature (or none)
- Good for Mind Control mechanic



InputActions

Store each action (move, jump, attack, etc) as an `InputAction` containing:

- Default and alternate `KeyCodes` (including Shift and Alt modifiers)
- Default controller button (create your own `ControllerButton` class!)

Mimic Unity's `Input.GetKey*` methods:

- `ActionState` enum: `Null`, `IsDown`, `IsHeld`, `IsUp`

Further considerations:

- Update `InputActions` before executing other scripts
- For mobile, have on-screen buttons change an `InputAction`'s `ActionState`
- To support the majority of controllers on desktop and mobile, check out the **InControl** plugin by Gallant Games

```
[Header("Input Actions")]
public InputAction navigateMenuUp = new InputAction();
public InputAction navigateMenuDown = new InputAction();
public InputAction navigateMenuLeft = new InputAction();
public InputAction navigateMenuRight = new InputAction();
public InputAction confirm = new InputAction();
public InputAction moveLeft = new InputAction();
public InputAction moveRight = new InputAction();
public InputAction jump = new InputAction();
public InputAction useWeapon = new InputAction();
public InputAction raiseShield = new InputAction();
public InputAction sprint = new InputAction();
public InputAction interact = new InputAction();

public InputAction toggleMainWindow = new InputAction();
public InputAction toggleDevConsoleWindow = new InputAction();
public InputAction escape = new InputAction();
public InputAction toggleChatBox = new InputAction();
public InputAction showNetworkScoreboard = new InputAction();
public InputAction toggleInventory = new InputAction();
public InputAction tabLeft = new InputAction();
public InputAction tabRight = new InputAction();
```

CreatureAI - Artificial Intelligence

AI controls a Creature:

- Sets the Creature's `MovementState`, `WeaponState`, and `ShieldState`

AI state **Patrolling**:

- Patrols back and forth
- `Physics2D.OverlapPoint` checks if we're going to bump into a wall; about-faces if so
- `Physics2D.Linecast` checks if Player's Creature is in view

AI state **Pursuing**:

- Jumps and/or walks to last-known position of Player's Creature
- Alert nearby Creatures?



So many creature types!

Predicament:

- Desire to have many different creature types (say 50)
- Most will be humanoid
- Do I create 50 different classes? 50 different scripts?
- What happens if I want to change every Creature's Transform hierarchy? Would I have to reorganise the hierarchies of all my Creature prefabs? (*Tallowmere* had this problem)

Solution:

- Create 1 generic, reusable Creature class (*without* creating smaller inherited classes!)
- Create a database to keep logical Creature data separate from any Transform hierarchy – *but how?*



Databases of things - Taking a key from Bungie's *Myth II: Soulblighter* (1998)



- *Myth II*'s monster data is kept separate from any code structure – Flexible, reusable
- Could something similar be used with Unity? Lists, enums, bools, floats, strings, Sprites

A screenshot of Fear, *Myth II*'s data editor

Creature Definitions

How about a simple **CreatureDefinition** component? Stored as prefabs in a folder:

- Acts as a **database** you can inspect and manipulate (faster than typing!)
- Data is kept separate from any complex Transform hierarchy
- Creature data is not bound to one creature type – uses simple bools and enums instead
- New definitions are as easy as pressing Ctrl+D

The screenshot displays the Unity Hierarchy and Inspector. The Hierarchy panel on the left shows a folder named 'CreatureDefinitions' containing several creature prefabs: Bloat, Conductor (selected), Feeler, FireMage, FlailKnight, Golem, Leaper, and Ogre. The Inspector panel on the right shows the settings for the 'Conductor' component, organized into several sections:

- Meta**: Name String ID is 'Creature_Conductor', Portrait Texture is 'None (Texture)'.
- Bools**: A list of 12 boolean properties, all currently unchecked: Becomes Confused With Low Health, Becomes Enraged With Low Health, Chained In Place, Cloaks With Stealth, Explodes On Death, Floats, Has Wings, Leaps Quickly To Target, Multiplies On Death, Throws Bombs, and Uses Tongue.
- Resistances**: A list of 10 resistance properties, each with a dropdown menu: Bleed Resistance (Very Susceptible), Cold Resistance (Susceptible), Fear Resistance (Average), Fire Resistance (Resistant), Knockback Resistance (Very Resistant), Lightning Resistance (Immune), Poison Resistance (Average), Polymorph Resistance (Average), Shrink Resistance (Average), Sleep Resistance (Average), and Stun Resistance (Average).
- Lists**: A list of 10 list properties: Spells (Size: 1, Element 0: Lightning (Spell)), Weapons, and Garments.
- Enums**: A list of 7 enum properties: Attack Speed (Slow), Attack Strength (Very Strong), Body Type (Humanoid), Eyesight (Average), Intelligence (Average), Movement Speed (Very Slow), Tankiness (Very Squishy), and Weight (Light).

Garments & GarmentDefinitions

- Archer_Glove
- Archer_Hair
- Archer_Pants
- Archer_Shirt
- Archer_Shoulderpad
- Esmerelda_Hat
- Esmerelda_Shirt
- LadyTallowmere_Glove
- LadyTallowmere_Hat
- LadyTallowmere_Shirt
- Leaper_Hair
- Leaper_Shirt

Similar setup:

- GarmentDefinition class that just holds data (stringID name, Sprites, GarmentType)
- GarmentDefinitions are stored as prefabs
- One prefab per GarmentDefinition

Creatures wear multiple Garments:

- A Garment loads a GarmentDefinition
- Places sprite(s) onto the Garment based on the definition's data
- Play dress-up!



Randomised Spells & Abilities

Defining **many spells** using **one class**:

- Leap, Poison, Pull, Polymorph, Fear, Stun, Charging, Shrinking, Teleporting, and more

- When creating **randomised loot**, game randomly chooses valid definitions to assign to a Weapon, Garment, or other item

- Can spells proc **OnCast**, **OnHit**, **OnCrit**, **OnKill**, **OnJump**? Are they **Passive**?

One proc type will be chosen by the **ItemGenerator**

- Make spells be compatible by anyone and anything where possible



Randomised Spells & Abilities (continued)

Area of Effect	
Aoe Requirement	Required
Aoe Can Be Frontal	<input checked="" type="checkbox"/>
Aoe Can Be Full Circle	<input checked="" type="checkbox"/>
Aoe Maximum Targets Can Be Unlimited	<input checked="" type="checkbox"/>
Min Maximum Aoe Targets	0
Max Maximum Aoe Targets	0
Chance	
Chance Requirement	Optional
Min Chance	<input type="range"/>
Max Chance	<input type="range"/>
Duration	
Duration Requirement	Required
Min Duration	1
Max Duration	4
Cooldown	
Cooldown Requirement	Required
Min Cooldown Seconds	<input type="range"/>
Max Cooldown Seconds	<input type="range"/>

Percentage or Integer	
Percentage Or Integer Requirement	Not Used
Can Be Percentage	<input type="checkbox"/>
Min Percentage	0
Max Percentage	0
Can Be Integer	<input type="checkbox"/>
Base Min Integer	0
Base Max Integer	0
Target Health Threshold	
Target Health Threshold Requirement	Optional
Threshold Can Check Owner	<input checked="" type="checkbox"/>
Threshold Can Check Enemy	<input checked="" type="checkbox"/>
Threshold Can Check Ally	<input type="checkbox"/>
Target Health Threshold Direction	Either
Min Target Health Threshold	<input type="range"/>
Max Target Health Threshold	<input type="range"/>

Spells can be applied to:

- Creature classes
- Creature races
- Weapons
- Garments
- Potions and temporary buffs
- Auras
- Room modifiers
- Enemy/Elite modifiers

Spell Modifiers keep items and gameplay fresh with randomness:

• Can the spell be single-target or AoE? Can it have a proc chance? Duration? Cooldown?

Percentage or integer for spell strength? Health threshold?

• Modifiers can be Required, Optional, or NotUsed depending on how the spell works

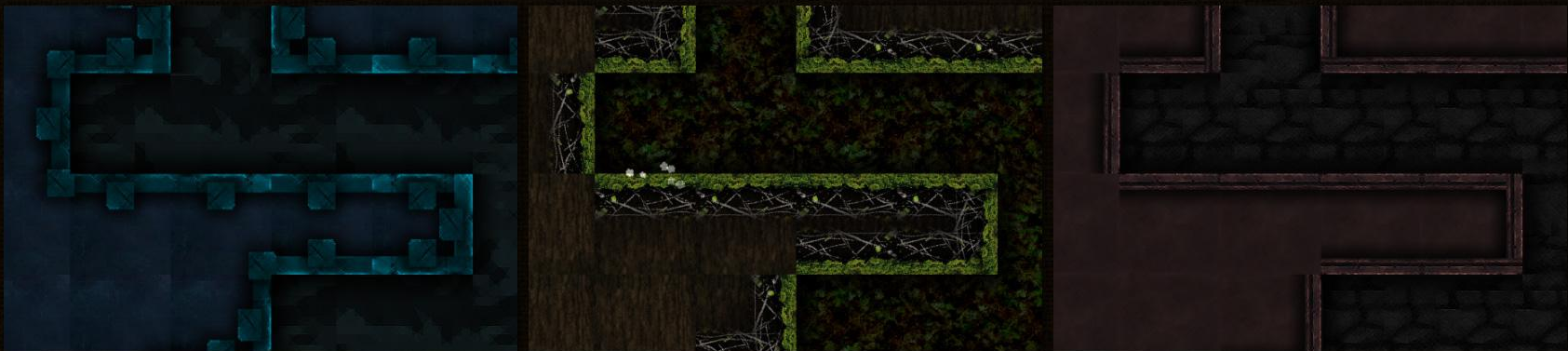
Rooms

Room creation:

- Each room needs X amount of spaces for enemies, items, and doors to spawn on
- Create `List<Vector2>` and traverse North, East, South, West randomly until enough spaces have been created

Players want cosmetic variations!

- RoomTheme definitions contain various sprites and cosmetic prefabs
- RoomCreator creates each dungeon room and applies the sprites from the definition



Seeds – Consistent Randomness

Create a `SeedFactory` class for use with **room generation** and **item generation**:

- Setting `Random.seed` lets `Random.Range` methods return the same results every time
- Good for testing and recreating the **same rooms** or **same items** over a network
- Note: `Random.seed` changes every time `Random.Range` is called

```
[Header("Seed Factory")]
public int initialSeed;
public int currentSeed;

[Header("Init Options")]
public bool useRandomSeed;
public int seedToUse;

public void Init () {
    if (useRandomSeed) {
        initialSeed = Random.seed;
    }
    else {
        initialSeed = seedToUse;
    }

    currentSeed = initialSeed;
}

public float GetRandomFloat (float minInclusive, float maxInclusive) {
    Random.seed = currentSeed;
    float result = Random.Range(minInclusive, maxInclusive);
    currentSeed = Random.seed;
    return result;
}

public int GetRandomInt (int minInclusive, int maxExclusive) {
    Random.seed = currentSeed;
    int result = Random.Range(minInclusive, maxExclusive);
    currentSeed = Random.seed;
    return result;
}

public int GetRandomSeed () {
    return GetRandomInt(-2147483648, 2147483647);
}
```

Sequencing – Eliminating Confusion



Simplify complex event sequences with a Sequencer:

- Break down each action you need to have happen
- Define each SequencerState as an enum:

```
enum OverworldToDungeonSequencerState {  
    ConfirmingNodeSelection = 0,  
    FadingToBlack = 1,  
    LoadingDungeon = 2,  
    PositioningPlayerCreatures = 3,  
    RevealingScene = 4  
}
```

- Sequencer switches through each state
- Takes sole charge of calling other classes' methods
- Use Timers for animations (like fading in/out)

Sequencers are also useful for:

- Network loading
- Setting up local co-op games

Networking – Hard but Achievable

Main goal is to sync each Creature's action and position:

- Creature is moving West, East, or Null at Position X,Y
- Creature started Attacking
- Creature is Jumping from Position X,Y
- Creature is now using Weapon X

What happens if a player joins mid-game?

- Send all current Creature data to the new player

Other considerations:

- Out-of-the-box sync vars are not enough – need to send your own data
- Use seeds for consistent room and item generation (just have to send an `int`)
- Let the host control the AI and damage numbers
- Do you force clients to wait for their own movement data to come back?



Networking – UNet vs TNet



Unity's UNet:

- Unity changes the API between versions
- Unity doesn't update the documentation
- Have to use Unity's servers for matchmaking (200 concurrent player limit)
- No automatic UPnP port opening
- If host leaves, game ends



Tasharen Entertainment's TNet:

- Full source code (no surprise changes!)
- Comes with a message-passing server you can host anywhere
- Automatic UPnP port opening
- If host leaves, someone else becomes new host automatically with zero fuss



User Interface Considerations

UI Scaling: Low resolutions? High resolutions?
Screen padding? 4:3? 16:9? Super widescreen?
Can user change the UI size?

Menus & Inventory: Reusable menu systems or
unique menu systems? Icons? Pagination?

Mobile Screens: Where are your thumbs?
Finger reach? Are buttons repositionable?

Keyboards, Controllers, Touchscreens:
What icons and symbols do you use to display a
Key or Button?

Local Co-op: How will the UI look for 4 players?



Supporting Multiple Languages

Support multiple languages from the get-go:

- Maintain a spreadsheet or database
- When you want to display text in-game, create a row
- Give each string an human-readable ID
- Ask translators to translate your strings, but leave the IDs alone

// String IDs	English
// Title Screen	
TitleScreen_Play	Play Game
TitleScreen_NewGame	New Game
TitleScreen_LoadGame	Load Game
TitleScreen_Options	Options
TitleScreen_Version	Version
TitleScreen_Exit	Exit Game

Using a language file:

- Export spreadsheet or database to a file you can import and parse (CSV, TSV, INI, TXT, JSON, XML, whatever works for you)
- Parse language file into a `Dictionary<string, string>` for the user's language
- Retrieve strings from the dictionary by ID, eg
`currentLanguageDictionary["TitleScreen_NewGame"]`

Singletons & Globals

If there's only going to be one instance of a class, I like creating a static link to it:

```
public static Global AudioManager;  
void Awake () { AudioManager.Global = this; }  
AudioManager.Global.PlaySoundEffect(soundEffectClip);
```

Lots of single-instance classes, allowing for quick access by adding `.Global`:

AudioManager

ChatBox

DialogueBox

HealthHud

LowHealthScreenOverlay

NetworkManager

ParticleManager

RoomInfoCornerHud

SeedManager

BigRoomText

CheatsManager

DungeonManager

ItemGenerator

LowHealthWarning

NetworkScoreboard

PlayerManager

ScreenShaker

SpriteManager

BlackVisionOverlay

CreatureHealthbarManager

FloatingCombatTextManager

LanguageManager

ModalManager

Overworld

RedHitCornerGlow

SceneTransitioner

WindowManager

CameraManager

CutsceneManager

GameSettings

LoadingScreen

NetworkDataLoader

OverworldNodeInfoBox

RedHitScreenOverlay

ScreenResolutionManager

Multi-platform Preprocessor `#if` Statements

Dealing with multiple platforms?

- Windows, macOS, Linux
- Steam
- DRM-Free
- iOS
- tvOS
- Android
- Wii U
- Xbox One
- PlayStation 4
- Dev-only builds

Execute code for certain platforms!

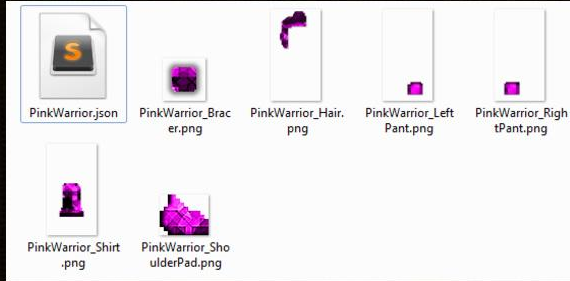
- Define your own Scripting Define Symbols under:

Edit > Project Settings > Player

```
#if STEAM
    // Do something with Steam
#elif WIIU
    // Do something with Wii U
#elif TOUCHSCREEN
    // Do something with mobile
#elif DEV
    // Add in some dev-only things
#else
    // Do the default thing
#endif
```

- Lets you retain one Unity project for multiple build targets

Mod Support



```
{  
  "CreatureName": "Pink Warrior",  
  "HairSpriteFilename": "PinkWarrior_Hair.png",  
  "HairBehindSpriteFilename": "PinkWarrior_HairBehind.png",  
  "HeadSpriteFilename": "PinkWarrior_Head.png",  
  "ShoulderPadSpriteFilename": "PinkWarrior_ShoulderPad.png",  
  "ShirtSpriteFilename": "PinkWarrior_Shirt.png",  
  "BracerSpriteFilename": "PinkWarrior_Bracer.png",  
  "RightPantSpriteFilename": "PinkWarrior_RightPant.png",  
  "LeftPantSpriteFilename": "PinkWarrior_LeftPant.png",  
  "ExclamationAudioClipFilename": "PinkWarrior_Exclamation.ogg",  
  "JumpAudioClipFilename": "PinkWarrior_Jump.ogg",  
  "TakeDamageAudioClipFilename": "PinkWarrior_TakeDamage.ogg",  
  "DeathAudioClipFilename": "PinkWarrior_Death.ogg"  
}
```

Let players customise your game:

- Image files and AudioClips in the StreamingAssets folder
- Text file containing path info
- Parse the text and import the assets to create GarmentDefinitions, WeaponDefinitions, CreatureDefinitions, RoomThemes, and more



Putting it all together

On the surface:

- Player creates character
- Player selects dungeon
- Player fights various creatures
- Player acquires shiny loot
- Player clears dungeon
- Rinse and repeat



Behind the scenes:

- Hundreds of different scripts making everything come together
- Creature hierarchies, animation systems, input handling, artificial intelligence, data organisation, randomised spells and items, countless states and methods, menu systems, networking, multiple platform targets
- Many little systems == One big game

Why do I do all this?

- So that people can have fun!

Parting Tips

Coding advice:

- Use `[Header("Header goes here")]` to group variables in the Inspector
- Unity's PlayerPrefs do not play nice with consoles. Create your own serializable classes instead
- UnityScript compiles slowly. Use C# instead

General advice:

- Start small – You'll grow as you go along
- Back up your work – Accidents happen
- Take action on your ideas – Experimenting gets results
- Seek and listen to feedback
- Take breaks – Come back when refreshed
- Be confident – You can do it!



Thank you!

Feel free to ask me anything.



Chris McFarland
Tallowmere

A copy of this presentation can be found at:
tallowmere2.com/nzgcd2016