Impossible Rescue.

Small Scale Video Game Development.

PDF Portfolio.

14th December 2019.

Amended for display purposes

Changes include removal of attribution table and file directory.

Links.

Build Video.

Stephen Warner

BIRMINGHAM CITY UNIVERSITY

Table of Contents

Impossible Rescue MVP.

The game Impossible Rescue has the following requirements set for the minimum viable product (MVP from now on):

- A single screen platform game.
- Basic player controls of left, right, jump, shoot, use, hack.
- Two weapon or ammunition types.
- Two enemy variations either in appearance, properties or behaviour.
- Progression mechanic requiring hacking to progress from each level.
- A life/death respawn system.
- Countdown timer for time-based levels.
- 12 balanced, fully tested playable levels.
- Coherent, consistent art style with usable clear UI.
- Target Visual which displays the art-style in-game.
- Fully detailed art style guide for the entire game.
- Working music and sound FX.

The Game Features.

For the end product:

- A female protagonist called Danielle along with her drone companion Shadow (Specialist Hard-linked Android for Duty in Offshore Wars).
- 2D movement using a console controller. (Left, Right, Jump, Fire and Interact).
- Collectable items which help the player to progress.
- Interactable objects which contain story elements and background information.
- Hackable terminals that are a requirement to progress the game.
- Multiple enemy types, both static and non-static variants.
- Three story acts with unique aesthetics with act 3 being a backtrack of acts 1 and 2.
- 12 levels with the last level per act having a countdown limiter.
- Prologue and end of act cut scenes to further the story
- The overall story has a “movie” time of 60 minutes.
- A drone companion called Shadow, which can hack, stun and boost the player.
- 3 story beats which detail parts of the journey through the game
- Music and Sound effects for all three acts.
- Moving Platforms.
- A stun-gun weapon.

Code Production.

I was again tasked with implementing the interactables, which include:

- Danielle hackable terminals
- Shadow hackable terminals
- Danielle Sound Effects (Walking, Jumping, Damage, Landing) ● Level Music (All music from levels 1 through 12).
- Server terminals which display story beats
- Danielle's weapon unlock
- The President's Daughter.

Aside from these assets I was also very involved with bug testing and resolution, along with the production and extensive testing of levels 1 through 8 minus level 4.

During the initial first weeks of development there were troubles with collision due to a "better" system being used which was different from how we were handling our collision detection during pre-production. This caused a significant amount of time to be wasted early on in development due to not knowing how this new system was supposed to function or how to interact with it as I was not in charge of implementing the CIntermediateLayer (This is a class and was called the physics layer very early in development and was later renamed). After having figured out the mechanics of the new collision detection using the Physics Editor bitwise masks/categories I was able to rapidly finish work on the terminals for Danielle and Shadow.

To pass the first stage of the production my team and I decided to use a band aid method to get the objects in the scene to interact with each other, i.e. Shadow's terminal to move a platform, Danielle's terminal to open a door, this was initially handled via CIntermediateLayer via pointers which get set on creation of the assets, which was a horrible way to do things, however for the initial development step it worked well. Later we held a short meeting to discuss how we were going to reform this system so that it uses OGMO (Level Editor) and the GCFramework creation parameters for including additional variables for creation, which could be linked together in code using the CProgressionHandler class and a few other classes like CHackable to determine what would control what and which assets would get linked. The variable we created in OGMO was able to handle strings and integers which made the system very flexible and convenient.

Aside from the initial teething problems which occurred due to physics adjustments, there was one other major aspect which I found to be irritating which was the audio management of Cocos2d. There is very little documentation on how to correctly use the two audio engines (Simple & Experimental) or what their primary differences are. This required a moderate amount of time and also consulting other teams in regard to their experiences with the two audio engines.

I would have preferred to have made the terminals to inherit from a base class which held various useful functions but due to time constraints I had to work very fast to add features or create entirely new objects which I was not aware of at the start of the production phase, because now there is a chunk of duplicated code which I would have rather avoided but due to time constrains was unable to remedy.

Cocos leaves a lot to be desired in the audio management section along with the documentation of the framework itself, even with the fairly substantial amount of research I had done on Cocos prior to the start of the course. We found ourselves guessing at functionality and discovering issues like memory leaking due to animations not being stopped before starting another animation, this wasn't discovered in pre-production likely due to the fact that there were very limited animations being called or played, as the only animation state change called was from the terminals, this time round we had the player constantly changing animation states which made the memory leak very obvious, however it took a few days to figure out where it was occurring and also what solution to use to alleviate or prevent it from being an issue in the future.

Our solution to this was to snap shot the memory and look at the differences, we eventually got lead to the RunAction - RunAnimation (Functions) calls, after discovering that we had simply added a line of code to call the StopAllActions on the sprite whenever a new RunAnimation/Loop was called. Those were the majority of the difficulties aside from communication problems between code/design/art teams which for the majority were resolved in time for the final build.

Since I was tasked with the implementation of a large portion of the levels from 1 to 8 and discontinued levels 9 through 12p1, I had to spend a large portion of time working with our level designer and ensuring the levels were playable and working to a decent standard, I unfortunately did not have time to proof check the levels 12p2 through 12p5 as they were not assigned to the code team for proofing.

Overall the project ended with a decent amount of success minus one error in the player animations for the pre-release build, the standalone build doesn't have the same issue.

Aside from the knowledge gained from the previous module and knowledge from my fellow programmers, the only aspect I had to research into was the audio management which was done using the Cocos2d-X Manual.

Code Annotations

```
void CWeaponChargePack::vPreCollision( b2Contact* p_cContact, bool isA )
{
    // get the fixtures
    b2Fixture* pourCollider = nullptr;
    b2Fixture* potherCollider = nullptr;
    p_cContact->SetEnabled( false );
    if ( isA )
    {
        pourCollider = p_cContact->GetFixtureA();
        potherCollider = p_cContact->GetFixtureB();
    }
    else
    {
        pourCollider = p_cContact->GetFixtureB();
        potherCollider = p_cContact->GetFixtureA();
    }
    // If the other collider has the mask data of the player.
    if ( potherCollider->GetFilterData().categoryBits & (1) )
    {
        m_pcPlayer = SafeCastToDerived<CPlayer*>( static_cast<CGCObjSpritePhysics*>(
            (isA ? p_cContact->GetFixtureB()->GetBody() : p_cContact->GetFixtureA()->GetBody())->GetUserData()) );
        m_pcPlayer->SetWeapon(true);
            CGCObjectManager::ObjectKill( this );
    }
}
```

The weapon pickup is super simple, if the player collides with the object SetWeapon to true on the player and finally destroy the weapon pickup

```
void CPresidentsDaughter::vPostCollision( b2Contact* p_cContact, bool b_isA )
{
    // If the other collider has the mask data of the Shadow.
    if ((b_isA ? p_cContact->GetFixtureB() : p_cContact->GetFixtureA())->GetFilterData().categoryBits & GetBit(ECollisionBit::EShadow))
    {
        // If shadow collides, kill the daughter then check if she was hackable, then if yes hack her.
        CGCObjectManager::ObjectKill(this);
        CHackable* pcHackable = GetHackable();
        if (pcHackable)
        {
            pcHackable->VHack();
        }
        // Will's method for sending shadow out of the scene.
        auto shadow = SafeCastToDerived<CShadow*>(static_cast<CGCObjSpritePhysics*>(
            (b_isA ? p_cContact->GetFixtureB()->GetBody() : p_cContact->GetFixtureA()->GetBody())->GetUserData()));
        if (shadow)
        {
            shadow->SendOutOfScene();
        }
    }
}
// This setter was created in anticipation of further logic which was not required in the final buil
void CPresidentsDaughter::RescueTheDaughter(bool bSaveDaughter)
{
    m_bIsSaved = bSaveDaughter;
}
void CPresidentsDaughter::VOnResourceAcquire()
{
    CPhysicsObject::VOnResourceAcquire();
    // Create the link between specific objects which have the same TerminalID number, in this case, if the daughter is hacked
    // then the object with the same ID will be activated/unlocked.
    CIntermediateLayer::GetInstance()->GetProgressionHandler()->CreateLink( GetFactoryCreationParams()->strExtraInfo, this );
}
```

This was a super simple class to handle the president's daughter. There was a lot of room for expanded functionality that was never utilized.

I made these functions in the CTerminal, CServerTerminal,
To save me time when I had to either show or hide all of these elements, it wasn't necessary but it made the process much faster and require less lines.

```
// Hide all Hacking progress bar elements
void CTerminal::HideProgressBar()
{
    m_pcCCSpHackingBarBackGround->SetVisible( false );
    m_pcCCSpHackingBarForeGround->SetVisible( false );
    m_pcGCSpriteHackingProgress->SetVisible( false );
    m_pcGCSpriteHackingCircle->SetVisible( false );
}

// Show all Hacking progress bar elements
void CTerminal::ShowProgressBar()
{
    m_pcCCSpHackingBarBackGround->SetVisible( true );
    m_pcCCSpHackingBarForeGround->SetVisible( true );
    m_pcGCSpriteHackingProgress->SetVisible( true );
    m_pcGCSpriteHackingCircle->SetVisible( true );
}
```

```
// A rather applicable use of a switch statement, based on the number specified in the ExtraData from OGMO,
// Set the visible storybeat animation accordingly, Beats to Variables are off by 1 due to frames starting from 00,
// and if 0 is in the extra data then default storybeat is shown.
void CServerTerminal::CreateBeat()
{
    const char* pszAnim_Beat00 = "StoryBeat01";
    const char* pszAnim_Beat01 = "StoryBeat02";
    const char* pszAnim_Beat02 = "StoryBeat03";
    const char* pszAnim_Beat03 = "StoryBeat04";
    const char* pszPlist = "TexturePacker/Sprites/StoryBeats/StoryBeats.plist";
    auto visibleSize = Director::getInstance()->getVisibleSize();
    m_pcGCSprStoryBeat = new CGCObjSprite();
    m_pcGCSprStoryBeat->CreateSprite( pszPlist );
    m_pcGCSprStoryBeat->GetSprite()->setLocalZOrder( 10 );
    m_pcGCSprStoryBeat->SetSpritePosition( Vec2( visibleSize.width / 2, visibleSize.height / 2 ) );
    m_pcGCSprStoryBeat->SetParent( IGCGameLayer::ActiveInstance() );

    switch ( m_iBeatNo )
    {
    case 1:
        m_pcGCSprStoryBeat->RunAnimation( pszPlist, pszAnim_Beat00 );
        m_pcGCSprStoryBeat->GetSprite()->setLocalZOrder( 10 );
        break;
    case 2:
        m_pcGCSprStoryBeat->RunAnimation( pszPlist, pszAnim_Beat01 );
        m_pcGCSprStoryBeat->GetSprite()->setLocalZOrder( 10 );
        break;
    case 3:
        m_pcGCSprStoryBeat->RunAnimation( pszPlist, pszAnim_Beat02 );
        m_pcGCSprStoryBeat->GetSprite()->setLocalZOrder( 10 );
        break;
    case 4:
        m_pcGCSprStoryBeat->RunAnimation( pszPlist, pszAnim_Beat03 );
        m_pcGCSprStoryBeat->GetSprite()->setLocalZOrder( 10 );
        break;
    default:
        break;
    }

    CIntermediateLayer::GetInstance(  )->VPauseTimer();
}
```

Above is a rather useful switch statement which controls what frame is displayed in the StoryBeat Sprite depending on what value is set in extra data in the OGMO CreationParams.

Below is an example of extra data being assigned to an integer called m_IBeatNo.

```
void CServerTerminal::Init()
{
    // Just a method to set the terminal to the neutral state on creation.
    // This only happens ONCE per terminal.
    m_pcPhysLayer = CIntermediateLayer::GetInstance();
    // Create any links to objects with the same TerminalIDs.
    CIntermediateLayer::GetInstance()->GetProgressionHandler()->CreateLink( GetFactoryCreationParams()->strExtraInfo, this );
    // Grab the extra info from OGMO,
    m_iBeatNo = GetIntDataFromExtraInfo( GetFactoryCreationParams()->strExtraInfo, "StoryBeat:" );
    m_pcPhysLayer->GetProgressionHandler()->CreateLink( GetFactoryCreationParams()->strExtraInfo, this );
    CreateProgressBar();
    m_bInit = true;

}
```

```
void CPlayer::vKillPlayer()
{
    m_pcAudioManager->play2d("Audio/PH_Danielle/PH_Danielle_Damaged.mp3", true, 0.5f);
    if (m_ePlayerState != EPS_Dead)
    {
    }
    m_bDirtyLabel = m_ePlayerState != EPS_Dead;
    m_ePlayerState = EPS_Dead;


}
```

This is how I did the player's audio effects listed previously.
The player makes use of the experimental audio engine from Cocos.
Whereas the levels use the simple audio engine from Cocos.

Final Build Video.

Most of my tasks were expanding upon previous work, meaning the only real new subject to learn was the audio management of Cocos2d-X

Bibliography

Cocos2d-x, 2019. *Advanced Audio Functionality.* [Online]
Available at: https://docs.cocos.com/cocos2d-x/manual/en/audio/advanced.html [Accessed November 2019].

Cocos2d-x, 2019. *Audio.* [Online]
Available at: https://docs.cocos.com/cocos2d-x/manual/en/audio/ [Accessed Novemember 2019].

Cocos2d-x, 2019. *Collisions.* [Online]
Available at: https://docs.cocos.com/cocos2d-x/manual/en/physics/collisions.html [Accessed November 2019].

Cocos2d-x, 2019. *Getting Started.* [Online]
Available at: https://docs.cocos.com/cocos2d-x/manual/en/audio/getting_started.html [Accessed November 2019].

Cocos2d-x, 2019. *Manual.* [Online]
Available at: https://docs.cocos.com/cocos2d-x/manual/en/ [Accessed November 2019].

Cocos2d-x, 2019. *Play Background music.* [Online]
Available at: https://docs.cocos.com/cocos2d-x/manual/en/audio/playing.html [Accessed November 2019].

Cocos2d-x, X. Y. S., 2018. *AudioEngine Class Reference.* [Online]
Available at: https://docs.cocos2d-x.org/apiref/cplusplus/V3.17/d0/d75/classcocos2d_1_1experimental_1_1_audio_engine.html [Accessed November 2019].