Play (Unity WebGL Version) on itch.io:
https://rileyelwell.itch.io/present-plunge
Demo Video: https://youtu.be/UYpTmPccrjQ
(If the demo video gives a playback error when clicking a YouTube link from an Adobe Acrobat PDF, reload the page to fix it!)

Futuregames Work Sample ~ Game Designer Education

Game: Present Plunge

Submitted By: Riley Elwell

Introduction

Short Description

Present Plunge is a 2D arcade-style game blending action and precision during the holiday season. In this endless runner type game, players catch presents to form stacks while avoiding falling anvils.

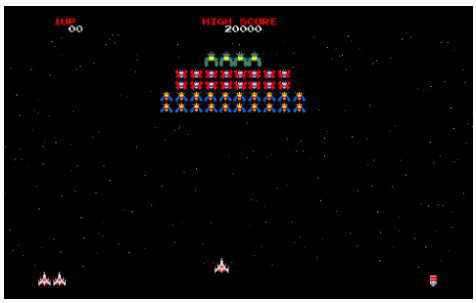
Theme

Every festive season, Santa embarks on his global sleigh journey, spreading holiday cheer by delivering presents to homes. Who can blame him for dropping a couple along the way? Besides, that's what elves are for! Step into the jolly shoes of one of Santa's little helpers and catch the presents that fall from his sleigh. But beware of the dangerous anvils plummeting from above! Make sure to be on your best behavior as you try to earn Santa's special rewards for swift grabs and impressive catches. Best of luck on this merry mission!

Inspiration

This game's main systems are inspired by the game jam's predetermined holiday theme and requirement to include anvils. I am deciding to create a 2D mini game because I am a fan of older arcade-style games and only have one week to develop it. Other games that inspired some design decisions include Galaga. Tetris, and Jamestown+.

The brief narrative is that an elf has been tasked with helping Santa catch and save the presents that fall from his sleigh. Unfortunately, there are anvils that also fall from his sleigh, and pose a threat to your helpful elf. This is inspired directly from the movie, Elf, as Buddy would do anything to help Santa deliver presents to the children each year.



Gameplay Concepts

Design Pillars

Precision

The game needs to make players feel like they need to focus to succeed, making it a challenge. They must feel the need to be precise in their strategies and actions in order to win.

Progression

The game needs to feel rewarding for players, giving them bonus points for achieving certain goals or milestones. It should provide the player with different content the longer the game continues in order to increase engagement.

Pandemonium (Chaotic Action)

The game should be very chaotic and require quick decision-making, challenging players each time they play. It should motivate players to improve their skills over time or adopt new strategies.

Core Gameplay

The main gameplay loop consists of the player stacking presents as they fall from the sky. They must place their presents on the ground to receive their score for each stack. To increase difficulty, if any part of a stack or the player is hit by an anvil, all of the current presents are reset.

The Player Catching/Stacking Presents

Win/Lose Conditions

The game will always only end when the player loses their three lives. A life is lost when the player or their current stack of presents is hit by a falling anvil. When the game ends, the score is displayed to the player before providing them options to restart, return to the main menu, or quit.

Although there is technically no “game-ending” winning condition, the player’s main objective is to score as many points as possible. This is achieved by stacking more and more presents, but different strategies may be applied. For example, bonus points are awarded to players for placing higher stacks or placing stacks quicker. While one person may try to create high stacks in a short amount of time, another person may take their time with smaller stacks to avoid falling anvils with greater ease.

Game Mechanics

Movement

The player should be able to move horizontally along the ground, but not vertically. It should feel smooth and responsive. The speed should be balanced between fast and slow to make a consistent difficulty.

Stacking

The player may automatically stack presents by catching them in their hands or on top of a previously caught present. The stack can have a maximum of ten presents to keep the screen space free. Hitboxes will be designed in order to achieve a higher level of precision from the player.

Placing

The player may place a stack down whenever they are holding at least one present using a keybind. This will activate the score calculator and they will be rewarded with the correct amount of points. This action can be performed anywhere on the map and has no cooldown.

Dash

The player should be able to quickly dash to the left or right every couple of seconds, activated by a keybind. This contributes to all three pillars as it adds to the chaos, forces the player to be precise in their movements, and motivates the player to be faster to gain higher rewards.

Development Process

Minimum Viable Product (MVP)

My first goal when tackling any project is to identify the most important components needed to have a game that functions. Then, I implement these components in order to produce my Minimum Viable Product (MVP). This means it may not look pretty, but the game is playable and contains the main features/mechanics I outlined from the start. After these tasks have been completed, then I can advance to making sprites, a clean user interface, smoother controls, gameplay tweaks, and a tutorial.

For Present Plunge I began with this list of must-haves (see video link below for MVP):

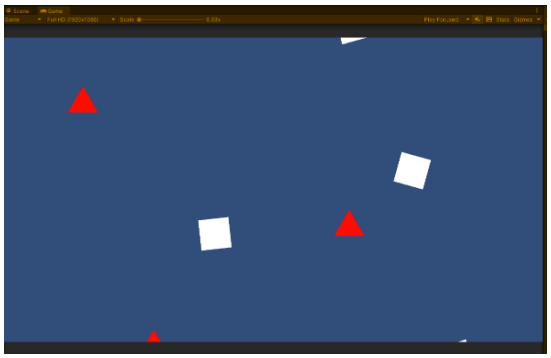
- Controllable player with horizontal movement
- Falling presents and anvils
- The ability to catch, stack, and place presents
- Conditions for winning and losing
- Scoring system

Object Spawning

When designing and developing a game, I always try to keep performance in mind. I know spawning/instantiating lots of objects at once in a Unity scene can overwhelm the processing quickly, so I thought of different ways to handle this. I opted for using an **object pool (see video links below)**, which essentially is an array that holds a predefined number of GameObjects that are spawned at the start of the scene being loaded. Then, the objects are set active and inactive in the scene whenever they are needed. This allows instantiation to only be called once, and all the script needs to do is update the recycled object's variables and positions.

```
// return the list of pooled presents (that are deactivated)
public GameObject GetPooledPresents() {
    for (int i = 0; i < _pooledPresents.Count; i++) {
        if (!_pooledPresents[i].activeInHierarchy) {
            return _pooledPresents[i];
        }
    }
    return null;
}
```

This function retrieves pooled present objects by iterating through a for loop to identify deactivated presents currently in the scene. It's designated as a public function to enable invocation from other classes via my singletons. These classes can then efficiently 'spawn' additional presents on the screen from the object pool, thereby optimizing performance.



[Object Pool Spawning Prototype]

[Object Pool Spawning Final Version]

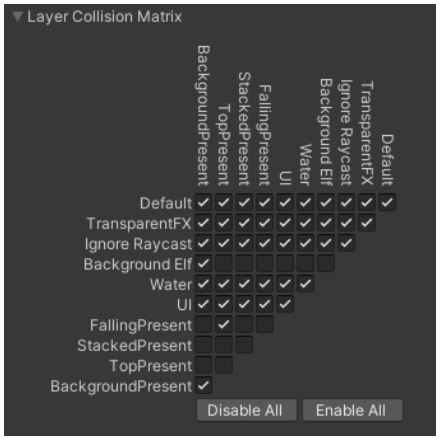
My object pool succeeded in spawning the anvils and presents, but there was an issue with the placed presents left on the screen. Although they were shifted to a background sprite, they still consumed memory as GameObjects. To address this, I introduced a "shadow elf" to clear them off the screen so they could be deactivated and re-added into the object pool. **This was a key design decision in stabilizing performance without sacrificing gameplay or narrative aspects.**

"Shadow Elf" Clearing Presents

Present Stacking

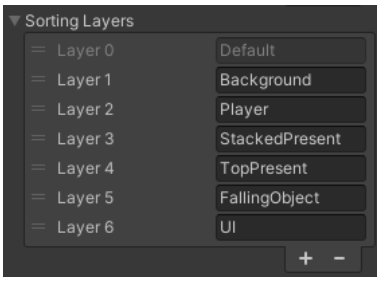
My first design consisted of squares and triangles with 2D colliders (see video links below), which checked collisions with other presents or the player via scripting. However, I soon realized the need to differentiate between falling presents, those forming the player's stack, and the stack's top. To achieve this, **I utilized tags, sorting layers, and collision layers**, ensuring each present stacked atop the previous one while maintaining proper sprite display. Additionally, my scripts ensured presents stacked only on the top of the stack, with anvils terminating the stack upon collision with any present or the player.

I initially planned to include a physics-based balancing system for the present stack, influenced by factors like stack height and player speed. However, I opted against it because it would introduce excessive chaos and hinder the player's ability to execute quick, precise movements like dashing. Additionally, this approach would detract from the core objective of building tall stacks quickly, instead emphasizing the struggle to maintain smaller stacks. Such a deviation from the design pillars would result in a dull and sluggish user experience.

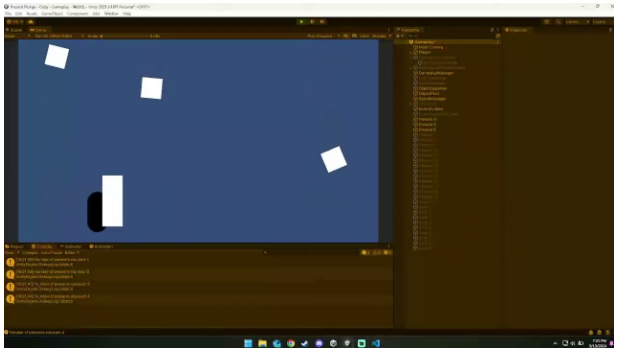


Collision Layer Matrix

Tags



Sorting Layers (Sprites)



[Present Stacking Prototype]

[Present Stacking Final Version]

Scoring System

A player should be motivated to increase their score, and therefore improve the skills needed to achieve this. In order to design this, the scoring should not just be a static counter for the number of stacked presents. This is why I decided to add bonus multipliers, based on the height of a stack and the time taken to place it down. The combination of these features in the scoring system **incentivises the player to improve**

their skills or adopt new strategies to achieve a higher score and makes the game more engaging the longer you play.

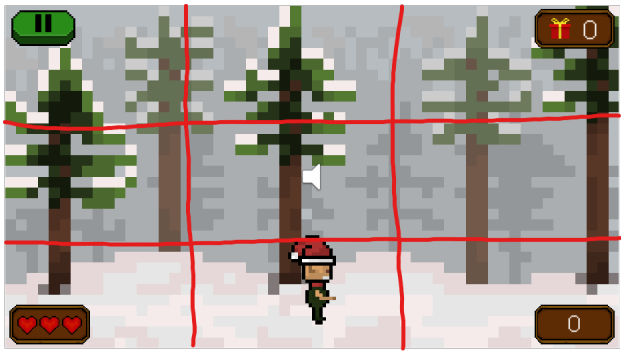
```
// this is called when the player places the presents down
public void UpdateTotalScore(int stackScore) {
    // reset the multipliers from the previous stack
    _stackTimeMultiplier = 1f;
    _stackHeightMultiplier = 1f;

    // update the height multiplier based on the stack's height
    if (stackScore == 10) {
        // print("Full stack bonus");
        _stackHeightMultiplier = _fullStackBonus;
    } else if (stackScore < 10 && stackScore >= 5) {
        // print("Half stack bonus");
        _stackHeightMultiplier = _halfStackBonus;
    }

    // update the time multiplier based on how long the stack has been held for
    float _totalStackTime = PresentStack.instance.GetStackTime();

    // if it is a full stack, there is a possible "fast" time bonus
    if (stackScore == 10) {
        if (_totalStackTime < 20f) {
            // print("Fastest time bonus");
            _stackTimeMultiplier = _fastestTimeBonus;
        } else if (_totalStackTime > 20f && _totalStackTime < 30f) {
            // print("Fast time bonus");
            _stackTimeMultiplier = _fastTimeBonus;
        }
    }

    // update the total score calculated with multipliers and display to the user
    _totalScoreValue += (int)(stackScore * _stackHeightMultiplier * _stackTimeMultiplier);
    _totalScoreText.text = $"{_totalScoreValue}";
}
```

This function calculates and updates the player's score. It determines whether multipliers should be applied by receiving the time and height for the currently placed stack via the singleton instance of the PresentStack class. Then, the UI text is updated to display the score to the player.

User Interface & User Experience (UI/UX)

My main goal is to keep a simple and clean user interface, maximizing screen space for gameplay while ensuring easy visibility. To achieve this, I created rectangles with large icons and light text on dark backgrounds. Following the Rule of Thirds, these are placed in the corners of the screen to **provide key information without distracting from gameplay.** All menus maintain a consistent style and aesthetic.

Keeping the 3 C's in mind for my player experience, I prioritized **clear instructions, responsive controls, and consistent gameplay actions.** Rather than a traditional tutorial, I opted for a written description of how to play that is displayed to the user at the beginning of the game. I chose this method because of the game's short length, limited ruleset, and simple premise.



Applying the Rule of Thirds

"How To Play" Screen (Tutorial)

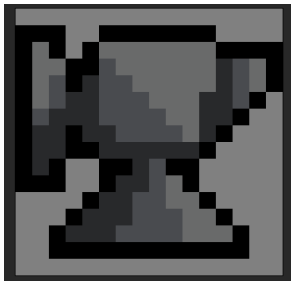
Pause Screen

Game Over Screen

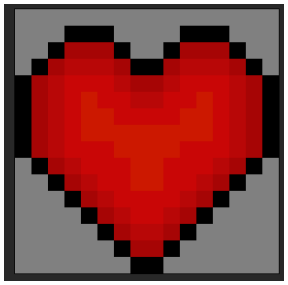
Achieving smooth and consistent controls required extensive adjustment of variables like speed, collision detection, and spawning rates. It was also tough finding a balance between a reasonable difficulty and the original chaotic feeling I wanted to capture. Playtesting with friends and game jammers provided valuable feedback, leading to adjustments to make the game easier initially and progressively harder as players stack more presents.

Art Assets

While my main focus is game design, I enjoy crafting my own assets to give my games a personal touch. I'm particularly drawn to pixel-art, which I've grasped the basics through online research and YouTube tutorials. Using Aseprite to make pixel-art for Present Plunge was an easy decision because of its 2D arcade-style gameplay. **My main goal with these assets was simplicity, emphasizing key features on each one for clarity.** Maintaining a consistent color scheme was also easy, as the Christmas theme guided my choices. Despite my lack of expertise in art and asset creation, I think they serve the game's purpose and scope well.



Anvil



Heart (Life)

Present

Elf (Player)

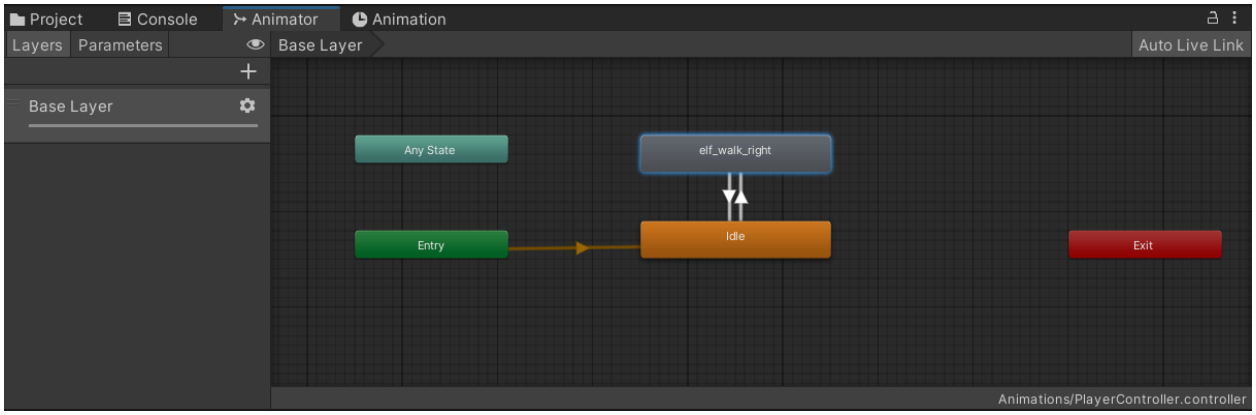
Animation

One extra feature I wanted to add to my game to give it more realism and character was the elf having a walk animation. Even though I have experimented with 3D animation before and know some of the basic principles, 2D animation is a completely different story. I struggled to develop a realistic walk cycle for the elf, and found it even more difficult to decide where to place certain pixels in each frame. After lots of trial and error I got something that wasn't perfect, but it resembled what I was going for.

After completing the necessary frames in Aseprite, I had to export it to Unity as a sprite sheet. I set up the animation with only two states, Idle and elf_walking_right. These states were hooked up to activate when a boolean indicated that the player was moving. I only needed one animation for walking to the right, because

I could simply flip the sprite in the other direction to achieve the desired result. I had to experiment with the speeds, exit times, and loops, but eventually I got a decent finished walking animation.

Elf Walking Sprite Sheet in Aseprite



Animation States for Player Controller in Unity

Closing Thoughts and Reflections

Documentation

Creating a comprehensive Game Design Document (GDD) for the first time proved immensely beneficial in organizing my thoughts and ensuring alignment with my original pillars and goals. Documenting my process with screenshots, video clips, and notes enabled me to review my progress from start to finish. This documentation not only clarified my decision-making process and reasoning at the time but also facilitated reflection on those decisions later on. Moreover, writing down my thoughts helped me stay focused and efficient in completing subsequent tasks by maintaining a clear goal in mind.

Testing & Iteration

The game development process is far from simple, and I found the testing and iterative phase to be the longest for me. Quick prototyping is key for assessing the fun factor of mechanics or systems, but achieving perfection takes time. Ensuring smooth, enjoyable gameplay with just the right level of challenge is

paramount for player satisfaction. I relied heavily on trial and error during testing, tweaking everything from stacking and movement to scoring and object spawning. It was a rewarding process to fine-tune numbers and settings until I struck the perfect gameplay balance. Overall, I learned to appreciate the iterative process and constantly looked for ways to enhance the game's feel through adjustments to its systems.

Future Improvements

If I were to revisit this project later, I'd aim to implement new features that align with my original design goals. Specifically, I'd like to introduce different variants of presents that provide the player with a greater score or powerups. These could enhance the dynamic nature of the player's abilities, such as enabling quicker dashes or adding jumps. I also envision creating distinct levels with fresh obstacles or challenges to maintain gameplay diversity. The last level in this new system could incorporate a boss fight to offer players a new objective and prevent boredom from solely chasing high scores. Lastly, I think exploring accessibility options such as a colorblind mode or difficulty settings would be important in creating a smoother player experience.

Final Reflection

In aspiring to become a game designer, this project emphasized the importance of all aspects of game development. While brainstorming ideas, I faced limitations in art skills and time constraints for coding, highlighting the necessity of considering the contributions of artists, programmers, and other team members. My goal with this game was to test and improve my skills in design, programming, and art, while creating a small, finished game. Despite recognizing areas where improvements could be made, such as better assets or longer gameplay, I am pleased with what I achieved as a solo developer within a short time period. This experience deepened my understanding of the challenges involved in game creation and fueled my eagerness to explore collaborative approaches.