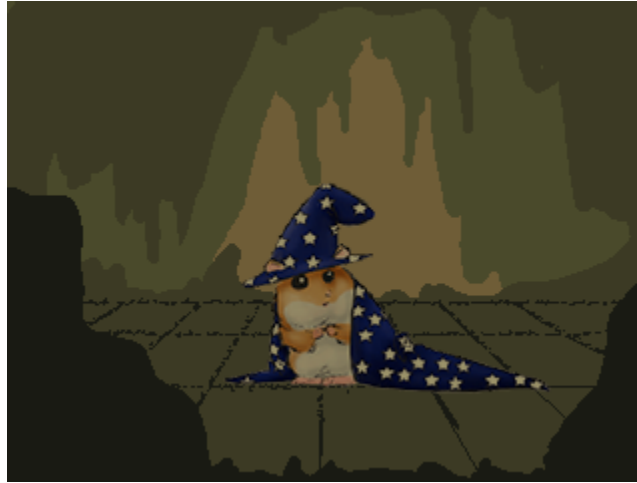


Pip's Quest

Design Documentation



Game Page (+ Executable): <https://benwakefield.itch.io/pips-quest>

Source Code: <https://github.com/wenbakefield/PipsQuest>

Demonstration: <https://youtu.be/PS9YvqMxFEI>

Slides: https://docs.google.com/presentation/d/10hi_Z0rfnZ4k9J0L1JVwdi_hHiV6pwDHDVZIW-eIWFA/edit?usp=sharing

High-Level Game Concept:

Pip's Quest is a roguelike, turn-based, puzzle game. The player plays as the hamster mage Pip, who embarks on a quest to defeat the demonic critters in his path, under the control of an evil snake. Each run is procedurally-generated based on a seed, which affects the enemies that spawn and the areas that they spawn in. The player must complete math puzzles by arranging runes from their bag in a spell, strategizing in order to deal damage to or defend against the critters in their path. The player is scored at the end of their run based on how much gold they have collected from defeated critters.

Procedural Systems:

1. Enemy Generation
2. Combat
 - a. Player Hand
 - b. Enemy Behavior
3. Loot
4. Area

Base Iteration

Enemy Generation

Species

Initially, there were 10 bosses, one of each species of enemies. The species included a frog, rat, bat, spider, meerkat, lizard, snake, rabbit, owl, and teacup pomeranian.

The boss is first generated, and then the regular enemies are generated based on the boss. Every three encounters, one or more of the enemies will be the same species as the boss. The chance of the first enemy to share species with the boss is 60%. The chance of the first enemy to be any of the other species is split evenly among the rest. If the first enemy does not share species with the boss, the chance of that happening for the second enemy is raised to 80%. If that does not happen again, the chance is raised to 100%. If the first enemy does share species with the boss, the chance of the next one sharing species is decreased to 40% instead. If that happens again, the chance becomes 20%.

Prototype

(Note: the probabilities and the number of species do not match with the first iteration)

Concept: A box with different colored cubes in it is used to simulate the generator of the species of enemies. The person who runs the simulation would pair a unique color with a unique species. The person would choose randomly from the box, and the cube's color would represent the species of the enemy/boss the player would encounter.

Set Up:

- A box
- 5 large cube, one of each color
- 50 small cube, 10 each color

Steps:

1. Choose a large cube randomly
2. Take out 5 small cubes of each color that does not match the large cube's
3. Choose a small cube randomly, and then put it back into the box.
4. If the cube's color matches with the large cube's, take out 3 cubes of that color. If the cube's color does not match, add in 3 cubes of each other color.
5. Choose a small cube randomly, and then put it back into the box again.
6. If the cube's color matches again, take out 2 cubes of that color again. If the result was different from the result of the last time, reset the number of cubes in the box to the number of cubes of step 2. If the cube's color does not match again, remove all cubes of the other colors.
7. Repeat step 5.

Elements & Weapons

Each enemy either belongs to a certain element or their weapon carries a certain elemental type depending on their species. There are four types of elements: fire, ice/water, earth, spark/electric.

The relationships between element/weapon and species are listed below:

Weapon: frog: club, rat: fork, bat: teeth, spider: knife, meerkat: goggles

Elemental: lizard, snake, rabbit, owl, teacup pomeranian

Prototype

Concept: The person running the simulation would pair a unique element with a unique suite in a standard deck of playing cards. The player would draw a card randomly.

Set Up: A deck of standard playing cards without the two jokers.

Steps:

1. Draw a card from the deck.
2. Put it back into the deck.
3. The suit of the drawn card represents the element of the enemy or its weapon.

Traits

Each enemy has a trait that modifies some of its stats.

The traits and their bonuses are listed below:

Shy (-1 attack, +2 defense), Brave (+1 attack), Reckless (+2 attack, take +1 damage), Cocky (-2 defense), Ripped (+1 attack), Cheerful (neutral), Lonely (-1 defense, -1 attack), Desperate (+1 attack, take +1 damage)

Prototype

Concept: The person running the simulation would pair a unique trait with a unique card from Ace(1) to 8 in a standard deck of playing cards. The player would draw a card randomly.

Set up: 8 standard playing cards (Ace to 8)

Steps:

1. Draw a card from the cards.
2. Put it back into the cards.
3. The number of the drawn card represents the corresponding trait of the enemy.

Health

Each enemy starts with their max health. Once their health decreases to or passes 0, they die.

Prototype

Concept: The player would draw cards randomly.

Set up: A pool of standard playing cards (contains 5 to 8 cards of all suites)

Steps:

1. Draw two cards from the pool.
2. Put them back into the cards.
3. The sum of the numbers of the drawn cards represents the base max health of the enemy,

Enemy Behavior

The enemy draws a card from a pool of cards with values ranging from 5 to 8. The suite functions the same, red for defense, black for offense. The enemy then attacks or defends based on the card they draw simultaneously as the player acts.

Loot

Enemies drop $4 + \text{trait score}$ (the sum of stat changes due to their trait) + level gold after they are defeated. There are consumables for purchase at the shop at the end of each area.

Spells

The player draws 5 cards from a pool of standard playing cards. The pool of cards consists of Ace (1) to 4 of all four suites, and two jokers. Each suite represents an element. Diamonds and hearts represent ice and earth, and those are the defensive elements, while clubs and spades represent fire and spark, and those are the offensive elements. The jokers are wild cards, which means that they can be used as any number, and cannot be used as the first card in a turn. The player can play however many cards they choose in a turn. However, they can only play a card with a value up or down 1 from the previous card they played. If the first card they played belongs to an offensive element, the player attacks. Otherwise, they defend. The value of either damage or block is determined by the sum of the values of all the cards the player played that turn. The player draws a new hand at the beginning of the next turn. The deck is shuffled.

Areas

3 Areas, divided into 3 sub areas in each large area. 3 enemies will spawn in each sub area, and there is a mini shop at the end of each sub area, and a large shop at the end of each large area. The player does not choose which area they spawn at and which area they would go to next. The boss resides in the end of the third large area.

Current Iteration

Species

The total number of species is reduced to 5. The species are: bat, bunny, bullfrog, spider, rat. The species of the boss is the snake.

Implementation

The species are determined with a pseudo random generator.

Elements/Weapons

Removed

Traits

The traits are edited. There are 50 traits, each trait modifies the base stats of the enemies. The total of the change by traits ranges from -3 to +3.

Implementation

The traits of the enemy are determined with a pseudo random generator, with the chance of a trait appearing following a normal distribution. This means that enemies are less likely to spawn with -3 or +3 traits, and most enemies are with neutral (net 0) traits.

Health

The pool of values from which the health of the enemies are drawn from is modified.

Implementation

A pseudo random generator pulls a number from the pool of values ranging from 7 to 9 twice. The sum of the values is the base max health of the enemy.

Spells

A pseudo random generator gives the player 5 runes without repetition. The rule of playing the runes stays the same. The chance of each element appearing is equal. There are two arcane runes, which function the same as the jokers. The player keeps their hand at the end of their turn. The players can reroll their entire hand once per turn (new).

Enemy Behavior

The pseudo random generator would decide whether the enemy would attack or defend. A pseudo random generator gives the enemy a value ranging from 2 to 4 if it is attacking, and then applies modifiers to that value. A pseudo random generator gives the enemy a value ranging from 5 to 8 if it is defending, and then applying modifiers to that value. The enemy acts simultaneously as the player.

Loot

If trait score is >2 or <-2 , it would be treated as 2 and -2 respectively during the gold drop calculation. There is a shop at the end of each area allowing the player to purchase 1 health per gold. The total number of gold left is the final score of the player when they win a run.

Area

5 Areas. 3 enemies would spawn in each area. A shop would appear after an area is cleared. The player gets to choose the next area they would like to visit after an area is cleared. Certain species have altered spawn rates in an area that is their habitat. The boss resides in a special area after all 5 areas are cleared. The current boss is a normal enemy with moderately higher stats.

Generative Code

1. Start with the initial concept. Who is it for, how do you know they want or need it?
 - a. Our main character inspiration!



- i.
 - b. Rogue-like gamers, PCG enthusiasts, card game players, people that can do addition, Professor Barney, people who like cute animals.
 - c. We wanted to offer an experience that combines many PCG elements into a cute, easy to understand game with an interesting math puzzle mechanic during battles.
 - d. Mobile/portable game style
2. Describe each design step you took to move it forward, what was the step, how did it change the project?
 - a. Our process was similar to a genetic algorithm, which makes sense since the typical iterative design process has a similar structure to a genetic algorithm.
 - b. We start by brainstorming ideas that we would like to implement into the game and list them out.
 - c. After creating this list, we rank the ideas based on the time it would take for us to implement it vs how beneficial it would be to the overall game experience.
 - i. This forms our fitness function.
 - d. We would then use the highest ranking idea and implement it into our prototype.
 - e. We would then playtest our updated prototype to see how it would work with the preexisting systems that we have implemented.
3. What was the fitness function you applied at each step in your process?
 - a. Initially we went from project ideation, by starting with the main theme that we would want regarding characters and scenery.
 - b. Once we had the idea that we wanted rodents/small animals within a forest, we moved onto thinking about systems and paper prototyping them to see how they would work.
 - i. This process loops between troubleshooting/brainstorming ideas and testing them out until we have a result that we are satisfied with.
 - c. From there, we moved on to implementing the coding portion of this onto a game engine as well as working on the art/animations portion of our game.
 - d. Once the text-based simulation of our combat is created, we would playtest the system to see what works best and if the game is balanced enough.
4. What heuristics did you use to 'unfold' the next step in your design? (Were these your patterns, or could you express them as patterns?)
 - a. Our heuristics were guided by the following design patterns:
 - i. [Procedural Questlines](#)

- ii. [Can I Get A Redo?](#)
 - iii. [Fight Like You Live](#)
 - iv. [And Now I Guess We Are Doing This](#)
 - v. [You can't have it all in this world](#)
 - vi. [Oh! That went unexpectedly well](#)
 - vii. [One of these days that's going to get you killed](#)
- b. We wanted to make sure that all of the systems within the game would work together to create a smoother gameplay.
 - c. We also wanted to make sure that it is possible for the player to win and progress through the rounds
 - d. Create motivation for the player to defeat enemies and progress throughout the rounds/levels.
 - e. Create a user friendly interface to boost user experience via visual effects and character sprites.
 - f. We followed the very loose definition of keeping a balanced game to make the gameplay more entertaining for the player.
 - i. It's dependent more on the amount of damage that the player deals and takes as well as the number of turns that it would take for the player to beat the enemy.
 - ii. We targeted an average of 10 turns, with the player taking the same amount of damage as they dealt at the end of the battle
 - iii. The generative pools are adjusted between areas (3 battles each) to attempt to adjust the enemy stats in order to achieve this target (adaptive difficulty)

Exercises

Exercise 4: Deck-Based Enemy Builder

Concept:

This is a deck based system that procedurally generates an enemy. The enemy has a unique species that either gives it a physical weapon bonus or a special elemental bonus. Additionally, each enemy also gains a randomly generated personality that can have a positive, neutral, or negative effect on their stats. Positive and negative personalities are more common than neutral ones, but this has not yet been properly balanced. We are planning on iterating this system for our main project.

Procedural Techniques:

1. Species

a. Physical Weapon Bonus

- i. Bullfrog
 1. Card: 1
 2. Weapon: Club
- ii. Rat
 1. Card: 3
 2. Weapon: Whip
- iii. Bat
 1. Card: 5
 2. Weapon: Syringe
- iv. Spider
 1. Card: 7
 2. Weapon: Throwing Knives
- v. Meerkat
 1. Card: 9
 2. Weapon: Night Vision Goggles

b. Special Elemental Bonus

- i. Lizard
 1. Card: 2
- ii. Snake
 1. Card: 4
- iii. Rabbit
 1. Card: 6
- iv. Owl
 1. Card: 8
- v. Pomeranian
 1. Card: 10

2. Personalities:

- i. Reserved (-1 attack, +2 defense)
- ii. Brave (+1 attack)
- iii. Reckless (+2 attack, take +1 damage)
- iv. Cocky (-2 defense)

- v. Buff (+1 attack, +1 defense)
- vi. Cheerful (neutral)
- vii. Lonely (-1 defense, -1 attack)
- viii. Desperate (+1 attack, take +1 damage)
- b. Weird (neutral)
- c. Aloof (-1 attack)

Justification:

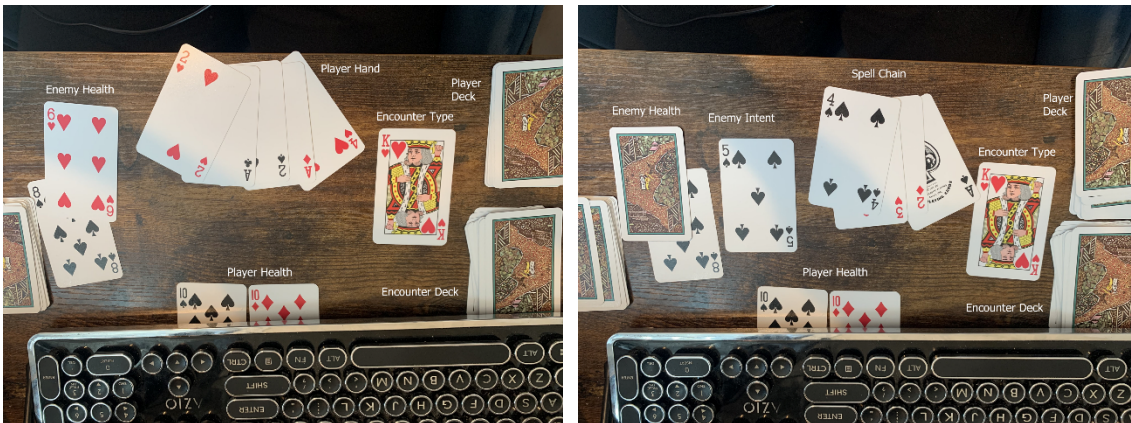
The combination of these procedural elements forms a unique experience in each encounter. Although some elements don't provide any stat changes, they work in tandem with the other elements to give each encounter more variety. Additionally, the player will never know what to expect, and some scenarios can subvert expectations. This is intended to secondarily provide a level of humor to the game, which adds to the character depth of each enemy.

Rules:

1. Elemental Types:
 - a. Offensive: Fire (hearts), Electricity (diamonds)
 - b. Defensive: Earth (clubs), Ice (spades)
2. Draw from 1-10 for enemy species, card suit will determine their special type if applicable
3. Roll 1-10 for enemy personality
4. Combat:
 - a. Players can play any cards in their hand
 - i. The first card played determines whether the player is defending or attacking
 - ii. Players can only chain cards with a value of one above or one below their first card
 - iii. The full hand is summed to determine the attack/defense power for that turn

Paper Prototyping:

Gameplay (will require pen and paper + 52 cards):



1. Draw two for player health from player deck (only aces and numerals)
2. Draw five for player hand from player deck
3. Draw one for encounter type from encounter deck (kings, queens, jacks)
 - a. If enemy, draw two from enemy deck for enemy health, draw one for enemy intent
 - b. If heal, add corresponding amount to player health
 - c. If item, +1 or -1 to any card with same suit in hand
4. If in combat, chain cards from hand together to form a spell

- a. The first card played determines whether the player is defending (black) or attacking (red)
 - b. Players can only chain cards with a value of one above or one below the previous card
 - c. The full hand is summed to determine the attack/defense power for that turn
 - d. Repeat until enemy or player loses all health
5. Keep any cards left in hand, clear the rest of the board, reshuffle all decks for next encounter

Procedural Elements:

1. Encounter Types:
 - a. King – Enemy
 - b. Queen – Heal
 - i. Different suits heal different amounts
 1. Hearts: 10
 2. Diamonds: 5
 3. Clubs: 3
 4. Spades: 1
 - c. Jack – Item (magical staff)
 - i. Different suits determine elemental type
 1. Hearts: Fire
 2. Diamonds: Electricity
 3. Clubs: Earth
 4. Spades: Ice
2. Enemy Encounter Builder (detailed in Exercise 4b)
3. Wild Card
 - a. Joker
 - b. Can be assigned any numerical value
 - c. Cannot be played first

Intent:

The procedural elements are all determined by a deck of cards, which makes the game very portable for the player. Additionally, the randomness of the encounter/combat system provides fairly interesting endless gameplay. However, in its current state it can be unfair and repetitive, and much has to be left to the imagination. This can be rectified with custom decks and an encounter log book rather than playing cards.

Playtesting Notes:

The game is playable in its current state, but as previously mentioned it needs to be balanced more effectively. There is currently no real sense of progression besides the player's hand. Additionally, unfair enemy encounters can catch the player off guard and make them feel like they have no control over the game at all, ending the run in a very dissatisfying way. These issues can be addressed with custom decks.

Procedural Content Effectiveness:

Overall, the procedural content forms the backbone of the game. It generates encounters and enemies very effectively, considering the source is only a deck of cards. However, it is not balanced, and in order to be more effective at giving the player a better experience it must be reworked.

Exercise 7: Procedural Sound Effects

For this exercise we focused on the procedural sound effect generation for spell casts. We want the sound effects to be based on how the player organizes a chain of cards from a randomly drawn hand. Thus, they will be different every time a player casts a spell. Additionally, the sound effect should be synchronous with the background music, matching its tempo and musical key.

We decided to use Python 3.9 with additional pygame, playsound, and numpy modules to prototype this system.

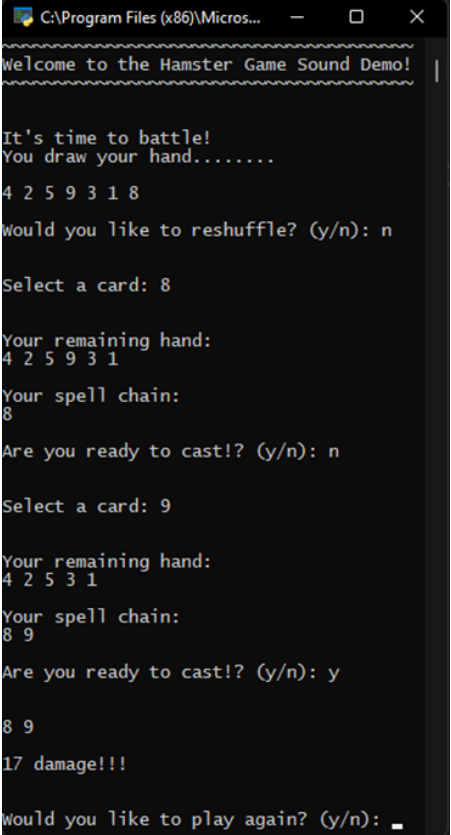
Encoding: List of integers, list of sound objects in minor pentatonic scale

Algorithm:

1. Randomly select 7 integers from a list of 1-10 integers
2. Display each integer in selection with a quarter note time delay
 - a. For each, play corresponding sound in list of sound objects using integer as index
3. Wait for user to chain a spell from their hand
4. Display each integer in finished spell with an eighth note time delay
 - a. For each, play corresponding sound in list of sound objects using integer as index
5. Display total damage and play a different sound based on level of damage

Representation: Cards, audio

This was a successful proof of concept for how the sound effects for spells should function in the final game. Due to the limitations of pygame, the music can become desynced from the sound effects if the user waits for varying amounts of time between inputs. However, this can be fixed with a timer or by waiting for the music to start over in the loop. It may also be interesting to have the user try to time their cast so that it syncs up better with the music, granting them a damage bonus.



```
C:\Program Files (x86)\Micros...
Welcome to the Hamster Game Sound Demo!

It's time to battle!
You draw your hand.....
4 2 5 9 3 1 8
Would you like to reshuffle? (y/n): n

Select a card: 8

Your remaining hand:
4 2 5 9 3 1
Your spell chain:
8
Are you ready to cast!? (y/n): n

Select a card: 9

Your remaining hand:
4 2 5 3 1
Your spell chain:
8 9
Are you ready to cast!? (y/n): y

8 9
17 damage!!!

Would you like to play again? (y/n): _
```

Exercise 8: Genetic Critter Generation

Source Code: <https://github.com/wenbakefield/HamsterEnemyGen>

Spreadsheet: https://github.com/wenbakefield/HamsterEnemyGen/raw/master/compiled_data.xlsx

→ As a Group: Create a genetic algorithm for any type of PCG.

- ◆ Generate enemies

- Select 2 numbers from a collection of 4, each with a certain probability of occurring. The base probability is 25% for each.
- Sum the two numbers to get the enemy health
- Select 1 trait from a collection of 10, each with a certain probability of occurring. The base probability is 10% for each.

→ As a Group: Generate a population of individuals using your PCG.

- ◆ Individual 1

- Health: 14
- Trait: Brave (+1 attack)

- ◆ Individual 2

- Health: 12
- Trait: Aloof (-1 Attack)

- ◆ Individual 3

- Health: 15
- Trait: Cocky (-2 Defense)

- ◆ Individual 4

- Health: 14
- Trait: Reserved (-1 attack, +2 Defense)

- ◆ Individual 5

- Health: 11
- Trait: Reserved (-1 attack, +2 Defense)

→ Individually: Design a fitness function for those individuals.

- ◆ Maximize enemies that will die in one hit

- The average damage that the player can deal in one turn was determined to be 8.5
- The fitness of the enemy will be how close its health is to 8.5. Lower values are better.
- The intent of the system is to have a generator that can be used to create enemies that will die in one hit or nearly one hit. This type of enemy would be useful in an early stage of the game to teach the player how to battle in a shorter amount of time, or how to craft their spell efficiently so as to defeat the enemy in one hit when it is possible.
- This selects for enemies with a very specific amount of health, and no health modifiers from traits.

→ Individually: Design a mutator for your PCG technique

- ◆ The parameters are the 4 numbers that determine the enemy health, as well as their corresponding probabilities.
- ◆ Lower numbers can be selected, and the lowest of those can have their probability increased while the highest of those can have theirs decreased.

- ◆ These changes will likely cause the enemy health to approach lower and finer health values. However, the ideal parameters to use to achieve maximum fitness will present themselves in testing.
- Individually: Generate a few generations of individuals using the above system you have documented.
 - ◆ Each generation usually is optimized and reaches the desired overall fitness level pretty quickly. However, sometimes it can get stuck in a loop, and the process must be restarted. I think this problem can be avoided as we add more parameters.
- As a Group: Compare the results of steps.
 - ◆ Simple fitness functions and complicated mutators had better results than vice versa. My fitness function simply compared the health of the enemy to a target value. My mutator took the values for health components and traits for the most fit individuals, calculated the probability of each occurring in the top set of individuals, and applied that directly as a mutator to the trait pool and health pool probabilities.
 - ◆ I saw a pattern during testing where using less individuals in your populations can provide results in less generations, but using more individuals takes longer to run and provides even better results, with a similar amount of generations to runs with 10 or 100 times less individuals.

