Timeless Archipelago

Creaking C⚙g Creations

Alex Toney, Allison Jett, Haoran Wang, Jarvis Huang, Sergey Maltsev, Pavan Nimmagadda, Wesley Brown

THE OHIO STATE UNIVERSITY

CSE5912

SUMMARY

Timeless Archipelago is an Action Role-Playing Game in the vein of already successful games, such as Diablo II and Path of Exile. However, we set our game apart by introducing an aging mechanic, which encourages players to start new playthroughs (which are styled as the "next generation"). The aging mechanic provides a new, critical choice to players – do they continue into old age as the game gets more difficult, or start a new playthrough?

CONTACT

Alex Toney
toney.77@buckeyemail.osu.edu
Allison Jett
jett.30@buckeyemail.osu.edu
Haoran Wang
wang.7371@buckeyemail.osu.edu
Jarvis Huang
huang.2089@buckeyemail.osu.edu
Sergey Maltsev
malltsev.1@buckeyemail.osu.edu
Pavan Nimmagadda
nimmagadda.8@buckeyemail.osu.edu
Wesley Brown
brown.5776@buckeyemail.osu.edu

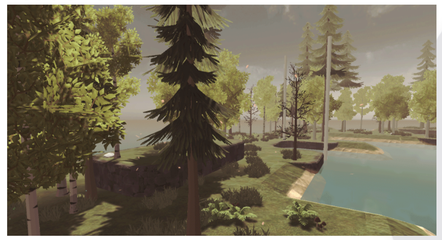
PROCEDURAL GENERATION

Overview

Taking inspiration from work done by Maxim Gumin as well as Karth & Smith (2018) [1], we modeled the procedural generation as a constraint satisfaction problem (CSP) which consists of variables and constraints. [2]

Our set of variables is a grid of squares, each of which can take on any terrain tile shape. When one tile is set, it constrains the possible tiles around it. The constraints are then propagated through the entire map. The process of selecting a tile and propagating constraints is repeated until all squares are filled in.

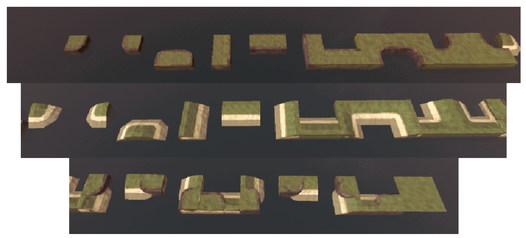
In order to exert more control over the procedural generation, we apply preprocessing steps and tile selection heuristics.



Tileset

We designed a tileset that allows us to generate islands of any size. Importantly, due to the design of our adjacency constraints, the tileset is non-Wang, as there are patterns that are disallowed even though the tile edges match.

In our tileset, 33 tiles are used per level of height. Most tiles are rotations of other tiles, so only 9 unique tile models were required. The first 33 tiles order by ID are pictured below.



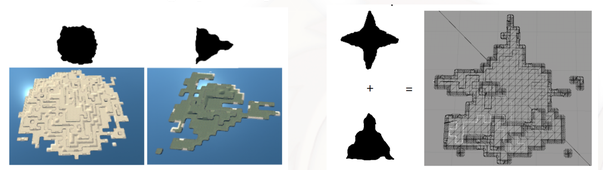
Tile IDs by row: 0-12, 13-24, 25-33

Each tile has a numerical ID. Tiles 0-32 are used to transition from water to land. Tile 33 is the first tile of the next range, and it is a flat land tile. Tiles 33-65 define the transition from one height level to the next. We can easily and dynamically add more tiles by reusing the information from tiles 33-65, with some simple calculations. For example, to generate an island that goes to height 5, the tileset is automatically expanded to 166.

Preprocessing

The procedural generation begins with the selection of one or more templates, which ensure that the overall shape resembles an island. The use of images for templates allows for easy template creation, as well as the ability to scale the templates to any size island. Templates can also be combined with each other to create new templates.

Since there are tiles that can block access to other tiles, we also ensure that there is at least one path to the boss teleporter by constructing a tree and then locking a path in place.



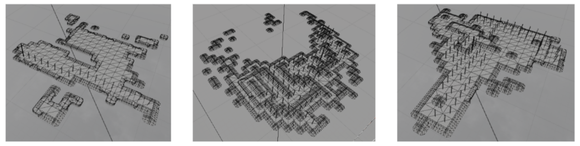
Constraint Satisfaction Algorithm

Once the island is guaranteed to be playable, the procedural generation proceeds as a constraint satisfaction problem. Our algorithm only uses adjacency constraints, although it is possible to define distant constraints. The algorithm is as follows:

- While there are undecided slots
  - Select the slot with the fewest values remaining
  - Select a tile from that slot's options using a tile heuristic
  - Propagate the possibilities to the rest of the slots using Arc Consistency Algorithm #3 (AC-3) [3]

Heuristics

We create heuristics that map the list of possible tiles in a slot to a list of weights. Our tile addressing system makes it easy to define functions that map IDs to weights. Additionally, we combine heuristics by multiplying the weights from different heuristic functions. Once the final weights are set, they are used to select a tile.



Very Flat + Hilly = Beachy Flatland

MODULARITY

All of our elements are modular and can be combined in a variety of ways.

Most things have a hierarchy of extension. i.e.: interface>abstract base > real base > specific object > etc.

This makes it very easy to add new instances of anything in the game (Items, Abilities, Enemies, Bosses).

ENEMY AI

A NavMesh is baked(created) after the island is generated. Each enemy uses a NavMeshAgent for pathfinding. Islands of different themes spawn enemies of different types and different colors. Each type of enemies has a unique field of vision to detect the target. Different types of enemies use different AI.

Bat/Spider moves randomly when player is not in the vision field, and follows and attacks player when it sees the player

Fungus/Plant sleeps/idles when player is not in the vision field, and attacks player when player gets close

Bee attacks the player only if the player hurts it. If one bee is killed, all other bees swarm and attack the player

		Theme				Stats				
		Grass	Swamp	Snow	Desert	HP	Attack	Mov. Range	Vis. Range	Mov. Speed	Atk. Speed
Bat	Melee				√	200	30	20	10	15	1
	Ranged		√			150	50				2
Bee	Blue			√		100	10	500	1000	50	0.6
	Green		√								
	Purple	√									
	Yellow	√									
Plant	Green		√			200	50	15	10	2.5	0.4 - 0.6
	Red	√									
	Yellow				√						
Fungus	Agaric Red	√				500	10	10	10	2.5	0.6
	Agaric Violet			√							
	Basic Brown				√						
	Basic White			√							
	Marchella Blue			√							
	Marchella Red		√								
	Marchella Violet		√								
Spider	Blue			√		150	50	30	15	10	0.4
	Purple	√									
	Red				√						

SAVING/LOADING

The player's data is saved to JSON files to allow continuation between different play sessions

- Inventory
- Equipment
- Age
- Afflictions
- Level
- Experience Points
- Skill Bar Layout
- Class Skill Tree

REFERENCES

[1] WaveFunctionCollapse is Constraint Solving in the Wild, Karth & Smith 2018
https://adamsmith.as/papers/wfc_is_constraint_solving_in_the_wild.pdf

[2] Artificial Intelligence: A Modern Approach, Chapter 6

[3] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99-118, 1977.