TerraShift Hexagon, Card, Turnbased, World-Simulation

Author: Otiose Entertainment (Leon Möhring)

Introduction

Thank you for purchasing this project. The base game was created during the 35th Ludum Dare competition, which is a 48 hours game jam. I highly suggest you to try out jams, because it is really impressing how much one can create in such a short amount of time. Anyways, this documentation doesn't cover every single line of code that went into the development, but instead teaches you important concepts necessary to create similar games. For the details you should dive into the code and read the comments so you don't get lost. But first read this paper to get an overview of the project. I hope you can learn very much by working with this game and have fun creating your own ones. If you haven't played the game yet, you should totally do that now!

The Game

In TerraShift you are a god! You have to change the world and use abilities to defend a village against droughts and beasts. After 30 years your score will be the largest number of houses the village once contained.

Instructions

The game is turnbased, each turn is equal to one year of in game time. You have 3 cards per turn and you must use 2 of them. There are cards to create new land and cards to change existing terrain. Some abilities cannot be placed on every land type.
After you finished the turn, the world changes like in a simulation. These changes follow specific rules. The village grows by itself, if it is located on green land.
You can try to learn the rules by yourself or read how it works below. Later on wolves and dragons spawn and destroy your houses. You can fight them by using certain abilities.

Controls

-Control with mouse and left mouse button (to select cards and land tiles)
-Pan view with middle or right mouse button

Project Structure

The whole game, in the current state, works with just 1 scene (Scenes/Main.unity).
Most of the GameObjects will get generated on start up, but some important objects exist in the scene hierarchy permanently.

Main Camera

The camera of the game is set to perspective mode. The attached CameraController script enables the player to pan the camera along the x-z plane by dragging with the middle or right mouse button. Alternatively, if bool drag is set to false, you can pan the view by pushing the mouse to the edge of the window. For further information on the CameraController, take a look at the comments in the script.
The skybox as well as the baked lighting settings of the scene can be changed under Window→Lighting.

Directional Light

The scene is lit up by the skybox (environment lighting) and this directional light. The light yellow color gives the scene a friendly atmosphere and soft shadows crate depth.

GridManager

The GridManager GameObject has the GridManager script attached to it. When you hit play, the script creates the HexGrid GameObject, which holds all of the hexagon tiles. More information on the hexagon grid can be found in the next chapter.

Selected

This GameObject represents the currently selected hexagon tile. It has a Mesh attached to it and is controlled by the GameManager script.

GameManager

This GameObject holds the core script of the game. Almost every action in the game is controlled by the GameManager, simulation rules, abilities of cards, beast Ais and Game Over conditions. For a game with a bigger scope one should consider splitting up some of the code sections into other scripts. If you want to understand how to the environment and village simulation work or how to write AI for characters on a hexagonal map, take a look at the corresponding chapters. The script also contains many comments, if you want to dive into the code.

MainCanvas

This canvas is always active. It will hold the cards during the game, which get instantiated by the GameManager script. It also contains the GameOver panel, which will get activated when the game is over and shows the final score.

EventSystem

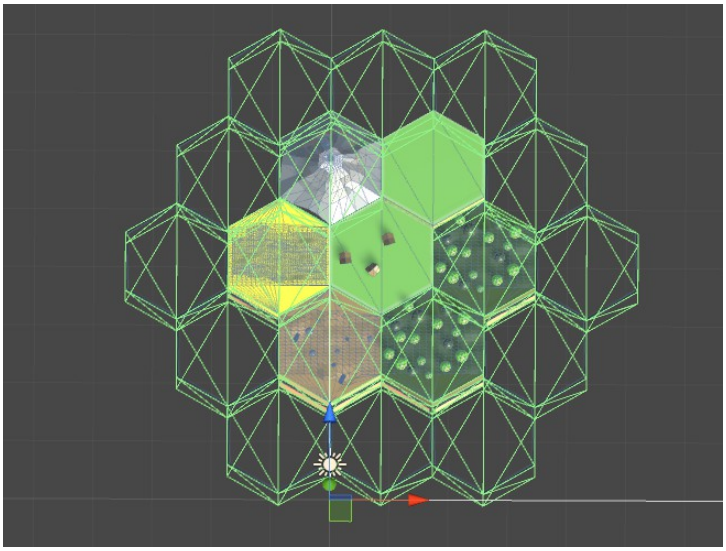
Unity's system to make the UI and Input work.

TurnCanvas

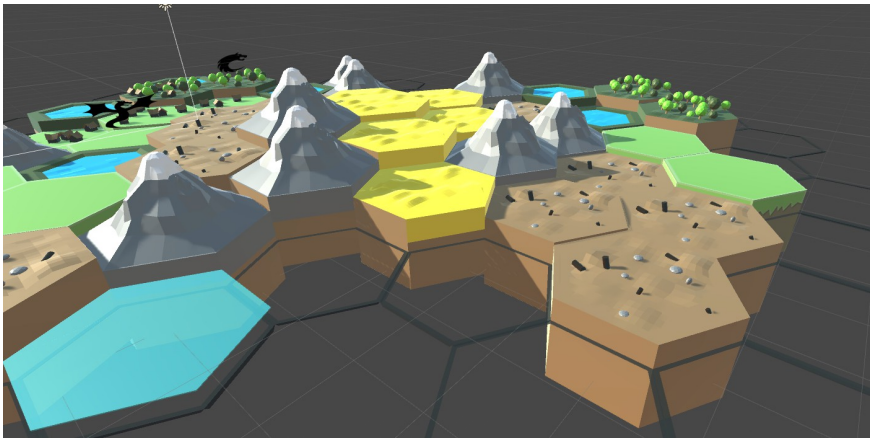
A canvas shown during the turn transitions to display the current year. When not in use, the alpha parameter of the CanvasGroup script is set to 0. This parameter is used to create fade in and out transitions.

Hexagon Map

The hexagon map is created by the GridManager script, which also contains important methods to change the map or get information. Using setHex() a specific tile at X Y can be changed to a new land type. Furthermore the getNeighbours method returns all the neighbors of a tile. The information of every tile is stored by the Hex script attached to each tile.



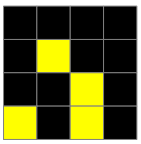
At the start the map gets filled with empty GameObjects on the correct tile positions. Then a predefined island of 7 tiles gets spawned using the setHex() method. The script also spawns 3 houses on the middle tile. To create areas for building new tiles using cards, the setHex method also creates “Void” tiles (a hexagonal grid model) on all emty neighboring tiles. This way it is assured that there will always be a ring of void tiles surrounding the island, even when the player adds new tiles.
To make them landscape more interesting and natural, the tiles Y coordinates are changed using Unity's PerlinNoise() function. It can generate a smooth 2D noise, which is perfect for height maps.



When adding a house, a new instance of the house model is created and positioned using the `Random.insideUnitCircle` variable. Then randomize the scale and rotation to create organic looking villages. Some overlapping houses are not a huge deal, but there is a maximum limit of 10 houses per tile.

Environment Simulation

One of the core game mechanics in TerraShift is the environment simulation. The survival of the village depends on the type of land. The simulation system used in TerraShift works like a more complex version of “Conway's Game of Life”.



In that game there are dead (black) and living (yellow) cells. Depending on the number of living neighbors the cell would either be changed to dead or alive once a turn. Some important differences in TerraShift are:
→ the hexagonal map (6 neighbors instead of 9)
→ more states a cell / tile can be in (desert, dirt, mountain, grass, forest, water instead of dead, alive)
→ no death of overpopulation (many forest neighbors are good, but too many living neighbors are bad)

Nature takes into account many different factors (and it would be a interesting to create an even better simulation system), but to simplify things TerraShift only calculates the “Green Level” of each tile. If a tile has less than 6 neighbors the missing ones aren't used in the calculation. The simulation adds up the green levels of all neighbors and divides the sum by the number of neighboring tiles. In other words, an average of all the neighbors green levels is calculated. Using that value and a set of rules the system determines the land type for this tile in the next turn and stores it. All changes are applied after all tiles have been processed to avoid affecting the algorithm with the land types of the next turn.

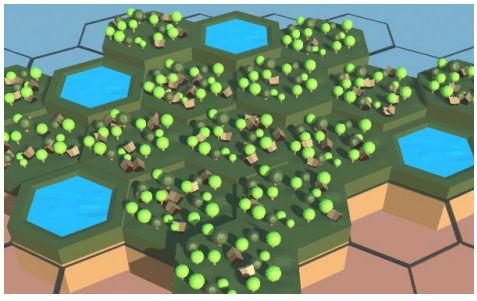
Green levels of all tiles are as follows:

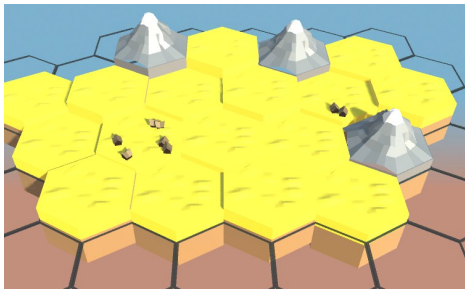
→ Desert (-2 Green)
→ Mountain (-1 Green)
→ Dirt (0 Green)
→ Grass (+1 Green)
→ Forest (+2 Green)
→ Water (+3 Green)

Rules for average value of a tile are:

→ Green <= -0.5 → Desert
→ Green <= 0.5 → Dirt
→ Green <= 1.5 → Grass
→ Green > 1.5 → Forest

There are some special rules though. Water and Mountain tiles cannot change their state because of their neighbors. Additionally, to slow down the system a tile can only change its type step by step. A forest surrounded by deserts cannot instantly become dirt, but first change to grass.
It turns out, that balancing a simulation system is a very tricky task. The environment shouldn't turn into a single extreme. It should have a certain stability to maintain diversity. The player has to have a chance to react. The special rules and the step-by-step technique do exactly that.





Card / Ability system

TerraShift is a slow, but strategic game. Many changes are made by the simulation algorithms, but that way the player has to use their limited actions intelligently and consider their effects. The player chooses two out of three cards and the tile on which to perform the card, but some abilities cannot be used on every land type. Because the cards are random, the player sometimes has to work with 3 negative cards and try to minimize the damage. Every game is different. There are 4 ability cards with different interactions:

→Meteor

Meteor changes grass / forest / dirt to mountain
Meteor changes mountain to dirt
Meteor changes water to grass
Meteor destroys all houses on the land

→Fire

Fire changes grass / dirt to desert
Fire changes forest to dirt
Fire destroys all houses on the land

→Rain

Rain changes desert to grass
Rain changes dirt to forest
Rain changes grass / forest to water

→Storm

Storm changes forest to dirt
Storm destroys half of the houses on the land

It is also important to note the interactions between the abilities and the village. Fire and storm will destroy houses on the tile. You also need abilities to fight the wolf and dragon.

Fighting beasts

Kill dragon with storm or meteor
Kill wolf with fire or meteor

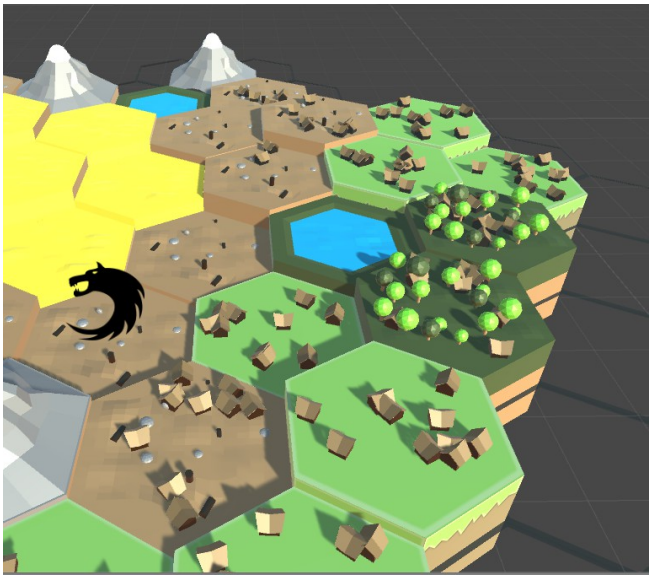
The rest of the cards are new tiles that can be placed next to existing ones. But to change old tiles you have to use ability cards. Again it is important to balance the mechanics. Rain is positive for land, fire is negative and the meteor allows to change the otherwise static water and mountain tiles. The card system works well with the turn-based nature of the game, but some form of deck-building would be nice. Even more depth could be achieved by making the abilities affect surrounding tiles and having long term effects. The meteor lands in a line of 3 tiles, the storm spreads, but cannot pass mountains and many more interactions are possible.

Village Simulation

The village is a core element of the game. After all the amount of houses is your final score. In the following the simulation system of the village is described. The actual house models and placing them is explained the “Hexagon Map” chapter.

Each house can create a new house on a neighbor tile, if the neighbors green level is high enough. This enables an exponential growth process, which is typical for populations. To limit the growth there can only be a maximum of 10 houses on each tile.

Houses cannot stand on water or mountains, but they don't get destroyed if the land changes between desert, dirt, grass or forest. On desert and dirt houses get destroyed over time. Desert prevents the spread mechanism and destroys 3 houses per turn. Dirt allows spreading, but also destroys 3 houses per turn.



To create an intelligent village behavior the spreading process follows some rules. The houses can only spread to neighboring tiles, so the simulation creates a list of possible neighbors (excluding mountains and water). Tiles with a green level <= 0 or the maximum amount of houses are also excluded. After that the list is sorted from highest green level to lowest. Houses will only get spread to the better half of possible neighbors, but will get spread equally on them. Example: A tile has 4 possible neighbors and 6 houses. Each of the two best possible neighbors will get 3 new houses.

Again all changes are stored first and applied after processing all tiles to not affect the simulation.

The results of these simple rules are pretty good, but one could think of different types of houses or farmlands and castles in the future.

Wolf and Dragon AI

The wolf and dragon are controlled similar, but they need to be killed with different abilities. At the moment the game only handles one instance of each. First of all the beats have to be spawned in the world. The chance for this to happen increases with each turn so it is very unlikely at the start of the game. The village has time to spread and build up. The spawn chance also increases with the time a wolf / dragon hasn't been on the map. The spawn position is a random land tile with no houses on it.

The beast then chooses a target tile. Because I didn't want to do path-finding in the AI at that point the beasts just choose a random tile with houses as the target and go there in an easy and straight path. If they find houses on their way, that is great, they will destroy them. They can move one tile per turn and destroy the houses on it. They can also cross water, mountains and even void because of missing path-finding. But because most abilities cannot be used on mountains, they now are a hiding point of beats and the bug turned into a feature.

Sadly the beasts are too weak at the current state of the game and fighting them is just a waste of time if the player is controlling the environment properly. But when balanced they are a critical game mechanic.