Tap Herder

Yeah, it’s a shitty name, but that’s what I have right now.

# Concept

This is a simple puzzle game in which the player attempts to optimize their points by “saving” a variety of creatures by guiding them to a destination without them being destroyed by a variety of obstacles.

Gameplay is on a hexagonal board, with each hex potentially being an empty (safe) space, an obstacle, or hazard. The player clicks on a hex in the game board, and the creatures behave according to their creature type (in the simplest example, by fleeing 1 space from the source of the tap).

|  |  |  |
| --- | --- | --- |
| C  C  C  C  C  C  Figure 1: Creature starting position | C  T  C  C  C  C  C  Figure 2: Tap “T” | C  T  C  C  C  C  Figure 3: Post-tap position |

The goal of the game is to optimize points by taking the fewest possible clicks to save the most creatures. A game instance (“board” or “level”) ends when all creatures are either saved or destroyed.

# Game Pieces

## Terrain

Conceptually, terrain may interact differently with different piece types. For example a “tar pit” hazard may be something a “frog” creature could leap, while a “snail” creature could not. Here are the initial types contemplated:

* **Hazard**: This is a space that will cause the destruction of a creature which enters it.
* **Passable Persistent**: This is a “normal” safe space.
* **Passable Destructable**: This is a safe space, that when tapped, is destroyed (becoming a hazard from that point on). Expectation is that destroying these pieces will add to the player’s score, as it increases the difficulty level.
* **Obstacle**: This is an impassable space, but one which does not cause the destruction of a creature which attempts to enter it.
* **Gate**: This space is passable in a limited number of directions (for example, only from left to right).
* **Void**: This is space off of the game board. Possibly this could change using a difficulty setting to be treated as an obstacle on easier modes and a hazard on harder modes.
* **Goal**: This is a destination space, which causes a creature to be removed from play (with the score being credited to the player).

## Creatures

There can be a variety of types to make the gameplay more interesting . Here are some of the ideas for creatures. Possibly these could be renamed to “fantasy” creature types.

* “**Dog**”: Moves 1 space along the ground.
* “**Snail**”: Moves 1 space along the ground, and treats “Obstacle” terrain as a hazard.
* “**Frog**”: Moves 1 space along the ground, and can leap occupied spaces and obstacles.
* “**Rabbit**”: Moves 2 spaces along the ground.
* “**Cat**”: Moves 1 space *towards* the tap.
* “**Wizard**”: Represents the player. Moves like Dog – game ends if destroyed.

## Anticreatures

These might function similarly to creatures, but are “bad” (no points for saving, and could potentially reduce your score by destroying creatures – player may gain points for destroying these).

* “**Wolf**”: Treat as “dog” for terrain purposes, but will destroy any creature it encounters, and moves *towards* the tap.
* “**Seeker**”: Moves towards the Wizard (regardless of distance). Destroys creatures it encounters.

# Scoring

All scores very tentative pending some testing.

|  |  |
| --- | --- |
| **Positive** | **Negative** |
| Creatures reaching a goal (+100) | Creatures dying (-50) |
| Time-based bonus (Up to 1000?) | Tap (-10) |
| Destroying a “Passable Destructable” terrain type (+25) |  |

# Achievements

Creating a specific pattern of residents.

Time achievement.

# Technical Notes

The core framework is built around the concept of a “game manager” which maintains relationships between the game itself, the source of input for commands, and the rendering of the game.

In this implementation, these classes fill those roles:

* **TapHerderGameManager** – Game manager, maintains the relationship between the game, the UI renderer and the source of input (“commands”) to the game.
* **TapHerderGameRenderer** – Displays the game state. In this implementation most of the work is actually done by the helper class **HexArrayRenderer**.
* **TapHerderInput** – The command input (mouse, in this case) for game commands.

These classes use some base classes which are unnecessary for the game (they could have just been written into the classes listed above), but which I put in place from a logical perspective in case the underlying game threading mechanic could be used elsewhere.

The base class is **TapHerder**, which starts the game manager . The main game loop is handled by the game manager.

The classes in the package **com.mele.games.utils.hexarray** implement a hex-based data structure. See the diagram below for an example. In this data structure the point (2, 1) does not exist, as it is off the map. However, all cells on the map can be referenced with X and Y coordinates where the X represents the column, and the Y represents the diagonal axis.

1

2

3

1

2

3

Classes in the package **com.mele.tapHerder.residents** represent creatures that live on the map and can be pushed around by the “tap” action.