CE810 - Game Design 2

Lab - Game Design Hack

Joseph Walton-Rivers & Piers Williams

Wednesday, 16 May 2018

University of Essex

Intro

CE810 game engine

- Remember we mentioned that we built you a game engine...
- well, here it is.

Limitations

- Games take place on a hex grid
- Games are turn-based
- No randomness

We originally designed it for Civilization style games, but it’s much more general than that.

Comparison

A number of you have **encountered** the GVGAI Framework.

GVGAI Framework	Our System
Custom VGDL files	Json standard based files
No ability to extend features	Ability to extend features
Slows down with additional rules	No such speed issues
Focuses on Interactions	Focuses on Rules

Game Engine

Key Parts

- A game has Entity Types, Resources, and Terrain
- Entity types have actions, costs and properties
- Resources and Terrain make up the maps
- Victory conditions tell you how to win (or lose)

Entity Types

- Used to **define** an Entity
- **Every** entity has a type
- Entity Types can **extend** other types
- Defines:
    - Graphics
    - Actions
    - Properties

Example: EntityType

```
{
"name": "abstract_civilian",
"properties": {
    "movement": 1,
    "health": 5,
    "attackRange": 1,
    "atkMelee": 1,
    "ter-grass": 1
},
"cost": {
"food": 10
},
```

Example: EntityType

```
"_actions": [
    "Move",
    "MeleeAttackAction",
    "Build[farm]",
    "BuildOnResource[lumber_mill:wood]",
    "BuildOnResource[gold_mine:gold]",
    "Build[marketplace]"
]
},
```

Entities

- Have an Entity Type
- Have **properties**
- Can perform 1 **Action** per turn

Actions

Actions

What an Entity can do

- 0 or more
- Parameterisable
- Inherited

Orders

Order

An order is **generated** when an Action is used on a **particular** location

- What an Entity **actually** does in its turn
- Used to **update** the game state
- Move Action $\rightarrow$ **multiple** possible Move Orders

Properties

- String $\rightarrow$ Integer mapping
- Used by default actions as well as custom ones
- **Two** sets per Entity
- Inherited

Terrain

Terrain defines the ground in the games

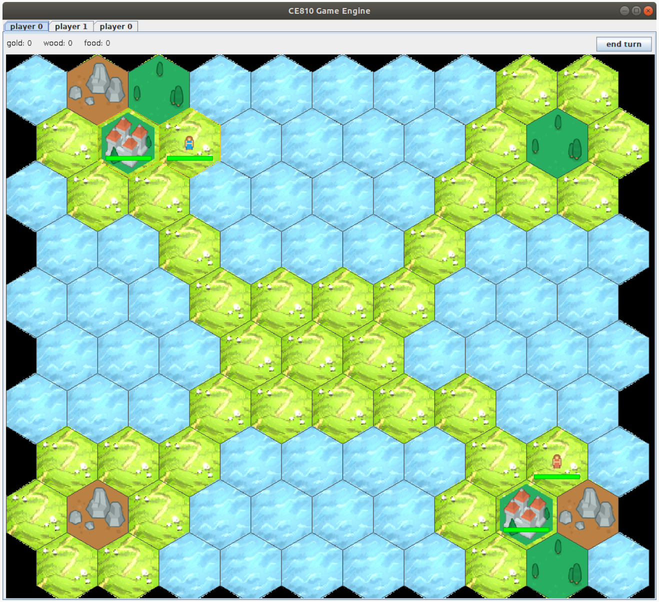
id	The name of this terrain type
image	The graphics path for drawing
requiredTags	Mapping of String $\rightarrow$ Integer.

Extensions

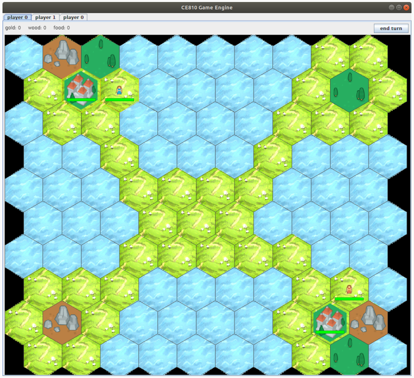
- The game is **extendible**
- You can **change** the json files **defining** the game
- You can **add** your own code
    - It will be detected on the classpath
    - Use the same way as the built in items
- You can add **new**:
    - Actions
    - Orders
    - AI
    - Victory Conditions

Examples

Medieval TBS

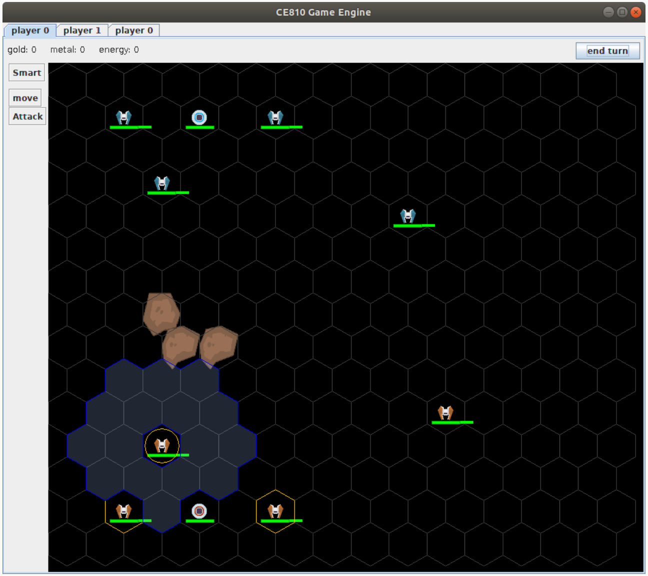


Medieval TBS

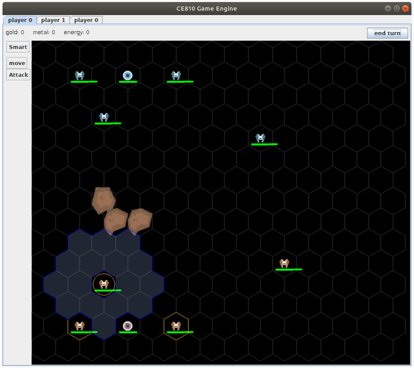


- Fairly conventional
- Build on resources for turnly income
- Civilians, archers, and knights

Transmission

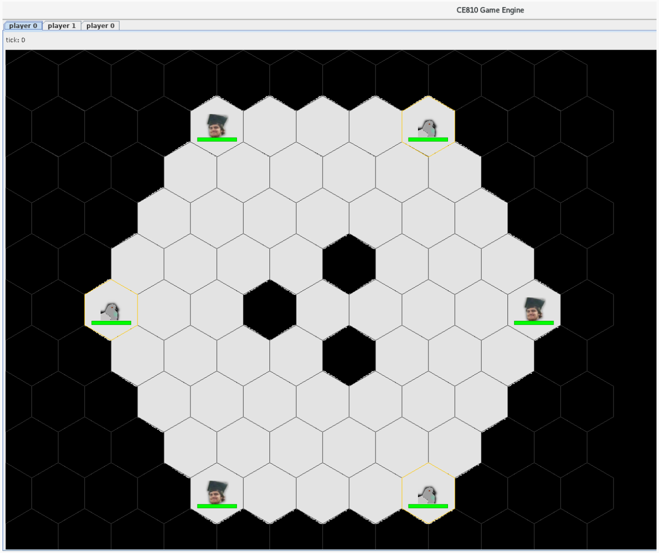


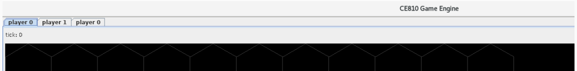
Transmission



- Global Game Jam 2018 Entry
- Space based TBS
- Units must stay **within** transmission range
- Can be **extended** with satellites
- Satellites can be **destroyed**

Hexxagon

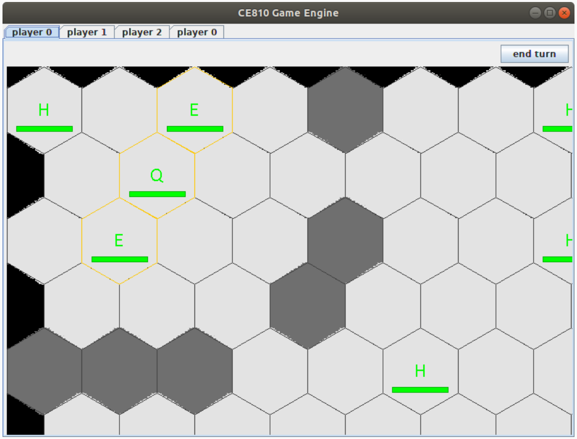

CE810 Game Engine
player 0
player 1
player 0
tick: 0


CE810 Game Engine
player 0
player 1
player 0
tick: 0

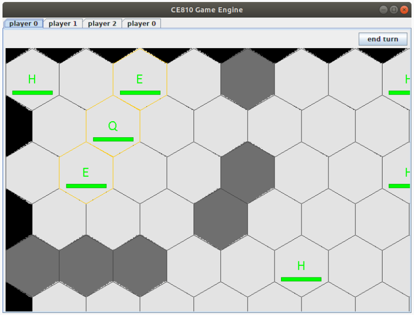
Hexxagon Entity Definition

```
{
  "name": "piece", // it's called 'piece'
  "properties": {
    "ter-playzone": 1, // it can 'walk' on
↪  playzone tiles
    "health": 1 // it has 1 health (things
↪  with no health die)
  },
  "_actions":[
    "Jump[tick]", // Jump Action (defined in
↪  Java)
    "Clone[tick]" // Clone action (defined in
↪  Java)
  ]
},
```

Aliens Versus Predators



Aliens Versus Predators



- 3 Teams
- Aliens
    - Queen Spawn Egg
    - Egg → FaceHugger
    - FaceHugger + Human → Incubator
    - Incubator → Alien
- Humans
- Predators

Your Turn

- This is what **we** did
- Demonstrates **some** of what can be achieved
- Your job is to make **interesting** games
    - Push the **limits** of the engine
    - Not a re-skinned TBS with **no** new mechanics
    - That have a reasonable design space for tuning
- Do not get hung up on graphics
    - Medieval game used a **single** set of assets designed for hexagons
    - Hexxagon and AVP used single colour tiles and basic images
    - Rules and interesting play are more **important**
    - Graphics serve to **distinguish** between different units

Design Patterns

Design Patterns

- Like programming patterns
- Many teams may have similar tasks to solve
- Some helpful patterns shown here

Movement Lock

Allow the player to only move one piece on their go

- Resource: time
- Only allow a move if the resource < current tick
- After a move is made, update the resource to tick + 1

Timers

You can define a timer by doing the following:

- Create an automatic action that performs the effect that you'd like to achieve.
- Set requirements to be "timeProperty $\geq$ timeRequired"
- Create an automatic action that generates 1 timeProperty
- Define the automatic actions as [generateAction, doneAction]