Super Action Kabanero

The tale of dungeons and guns

Documentation

Version 0.1

Software structure

See http://mureke.gitlab.io/kabanero/annotated.html for documentation and more in-depth diagrams.

Every object into the Super Action Kabanero world is referred to Tree structure as a `Node`. This helps with development and updating for each frame. Each `node` location is dependent of all it's parent `nodes` transformations. `Nodes` have different behaviours which determine how the `Node` behaves in each game frame, such as `BulletBehaviour` which is attracted to each bullet `Node` that is generated. Every `Node` can send `Event` to the `messagePublisher`, afterwards it sends these to their receivers.

The main game loop for each frame happens in `App.hpp` which is loaded from `Main.cpp` file. In the App, services are initialized. Services are common objects that needs to be accessible from different parts of the code. Services used are:

- **Logger**
  - Mainly for debugging reasons made class that prints information about the game to the console.

- **ResourceManager**
  - Loads different files like audio clips, fonts and texture atlases.
  - Keeps resources centrally managed and easily accessible.

- **MessagePublisher**
  - Takes care of messages that are sent to message subscribers. Every event has specific information, usually stored in enums.

    AudioInputEvent
    Handles audio action information and sends it to the `AudioPlayer`.

    GameInputEvent
    Handles user input information and sends it to the game.
    `InputTranslator` maps user input keys to specific Events.

CollisionEvent
Collision information is relevant when checking for bullet and player or monster collisions for example.

For inputs SFML's events are polled in app's `run()` loop. These are translated to `GameInputEvents` by `InputTranslator`. Events are polled and processed each frame.

Currently the whole application consists of two scenes that are `GameScene` and `MenuScene`. Menu is used to choose between single player and different multiplayer modes. After that game initializes and loads needed components and the game starts.

Description of software logic

For map generation we have used cellular automata to generate cave-like structures. The basic idea is to fill the first map randomly, then repeatedly create new maps using a modified set of rules based on the Game of Life. Each iteration has two thresholds, if a tile has less "alive" neighbors than the first threshold or more "alive" thresholds than the second it itself becomes alive. This forms cave-like structures, with medium-sized open areas as the under-a-threshold rule prevents formations with too many clustered dead tiles from forming. After generating a cave, some basic wall-structures are stamped on top of it. The map generation always provides the same result for some given seed.

We are using Box2D library which is an open source C++ engine for simulating rigid bodies in 2D. Box2D is developed by Erin Catto and has the zlib license. In this particular project mainly we are using this library to handle the collisions with the objects.

Instructions

Project folder structure

```
.
├── doc         # Doxygen config
├── include           # C++ Header files
├── lib         # External libraries
├── raw_assets  # Unbuilt assets
├── resources   # Game resources
├── src         # C++ Source files
├── test        # Unit testing
└── tools       # External tools
```

External dependencies

SFML is required to build and run the software. Other dependencies are included in the repository.

Java is required to build sprite atlases.

Building

Building process is tested with Ubuntu 16.04
Clone and setup the repository:

```
$ git clone git@git.niksula.hut.fi:elec-a7150/mbombers2.git
$ cd mbombers2
$ git pull && git submodule update --init --recursive
```

Build the game with following commands:

```
$ ./build.sh assets     # builds assets
$ ./build.sh run        # builds and runs the game
```

Runnable executable can be found in `build/bin` folder.

Enable sound for the best experience.

Using the software

Controls

In the main menu use `P1 Up` and `P1 Down` to move between choices and `P1 Right` to select. You can return to the main menu with Escape.

Keybindings

W	P1 Up	Y	P2 Up	Up	P3 Up	Numpad5	P4 Up
A	P1 Left	G	P2 Left	Left	P3 Left	Numpad1	P4 Left
S	P1 Down	H	P2 Down	Down	P3 Down	Numpad2	P4 Down
D	P1 Right	J	P2 Right	Right	P3 Right	Numpad3	P4 Right
1	P1 Fire	5	P2 Fire	Delete	P3 Fire	Numpad7	P4 Fire
2	P1 Strafe	6	P2 Strafe	End	P3 Strafe	Numpad8	P4 Strafe
3	P1 Bomb	7	P2 Bomb	PgDown	P3 Bomb	Numpad9	P4 Bomb

Gameplay

Super Action Kabanero is a multiplayer deathmatch focused game. The objective is to survive and be the last man standing in the battlefield. Be aware of your enemies as well as numerous monsters that are patrolling around the level. Each player has 3 lives.

- ❖ You can use your weapons and bombs to destroy terrain creating new passages or tactical hiding places. Use this to gain an advantage.
- ❖ Bombs can be used in mining as well as in battle. You can shoot bombs to push them near your opponent. Bombs explode automatically after short delay.
- ❖ There are some weapon pickups hidden in the level. Note that to find the best weapons you are required to do some digging.

Testing

Game mechanics were vigorously tested during development sessions. This gave multiple good game design ideas that were implemented later.

All of the merge requests are automatically tested using Gitlab's CI toolset. Every merge request that was accepted was required to have a successful build status. Major parts of the engine were unit tested with Catch framework.

Note that the unit tests are commented out for GCC support.

Known bugs and problems

- ❖ Keyboards with limiter rollover may cause 'locking' of keys especially with multiple players. Multiple keyboards are recommended for multiplayer mode.
- ❖ Then changing scenes (from menu to game and vice-versa) memory allocated for collision data is not handled correctly causing a memory leak.
- ❖ Sprite rendering is sub-optimal causing slowdown when a large number of sprites are drawn. This mostly affects the 4 player mode.
- ❖ With accurate timing, bullets may rebound back to the player if player shoots immediately after dropping a bomb.
- ❖ Two players may spawn to the same location.
- ❖ Enemies may get stuck if they come too close to a wall.

Work log

Weekly time estimates

	Toni	Juuso	Tuomas	Max
- Nov 7	20	20		< 5
Nov 7 - Nov 13	5	10		< 5

Nov 14 - Nov 20	15	15		< 5
Nov 21 - Nov 27	15	25		< 10
Nov 28 - Dec 4	15	20		< 10
Dec 5 - Dec 11	35	20		< 5
Dec 12 - Dec 16	25	20		< 5

Main Responsibilities

- Toni
  - Resource management development
  - Audio coding and design
  - Menu and hud coding
  - Graphical effects
  - Enemy behaviors
  - Physics integration
  - Project management and merge requests

- Tuomas
  - Level generation and related algorithms
  - Line of sight and Fog of War
  - Player moving and shooting (1st iteration)
  - Physics integration
  - Different guns and pick-ups

- Juuso
  - Engine core design and development
  - Rendering
  - Optimization (engine and rendering)
  - Collections
  - Project management and merge requests

- Max
  - Testing and issue management
  - QA evaluation
  - Input translation
  - Weapon logic (1st iteration)