SYSC 3303 WINTER 2014

TERM PROJECT

The goal of the term project is to develop the *Carleton University version of the* Bomberman game. There are many versions of Bomberman. **OUR** version will use a subset of the rules posted here as well as a few modifications noted below.

> *Important: For Bomberman aficionados , you are free to add rules and features OVER AND ABOVE those described in this document, but you must not add a rule or feature that contradicts this document.*

Our user objectives – within the limits of the term - are:

- We will just have one floor, not 50, although the design and implementation should both be easily extended to multiple floors.
    - A floor is a matrix of squares.
    - Upon startup, squares are initialized as either empty or holding a box and/or containing the exit door. Additionally, one [1]square will contain one power-up.
- The game begins when the first player hits the START_GAME control. Until that time, other players can join
    - The floor is initialized and a random number of enemies are launched.
- The game ends when a player exits the door.
    - The door is initially hidden, either not visible or hidden under a box..
    - The door is revealed by a player landing on a square (if the door is simply not visible) or by a player blowing up a box that is covering the door.
    - A player may exit the door only once all enemies have been terminated.
- A player has one life
    - A player dies if it is touched by an enemy or another player, or it is within the explosion range of a bomb
    - A corollary of the previous point is that a square holds at most one player.
- A player has access to bombs
    - Players are initially given one bomb.
    - A power-up increases both the number of bombs that a player can hold and the explosions range of the bombs.
    - The explosion range begins at 1 square (not including diagonals).
    - Bombs detonate after 2 seconds (to allow the player to move away). Until that bomb has exploded, no further bombs can be deployed (if your limit is still at one).

[1] Subject to Change: The official rule is for one power-up per floor. Because we are only doing one floor, we may change this rule to allow for a few power-ups on our floor. However, if we have multiple players, should we have one power-up per player?

- We will support multiple players, although it is fine to put a finite limit on the number of players supported at one time.
    - To begin, set the limit as two. Maybe in Milestone 3, we will up it to four (so the whole team can play together).
    - We will not provide player login, names, scoreboards, etc.
    - In the third milestone, we are going to aim for an autonomous player (i.e. with AI).

Our technical requirements[2]

- The system will all be written in Java, in the hope that you'll continue and extend your work, if you enjoyed making this system.
- All properties should be configurable, such as the dimensions of the floor, the number of players permitted, the maximum number of enemies deployed, the update rate of the game state, the speed of the enemies, and so on.
- The system shall be built as a distributed system using a client/server model that implements the Model-View-Controller pattern
    - The server shall contain all game-logic
    - The player clients shall contain a view of the game and the player controls
    - A spectator client shall contain only a view of the game.
    - The system shall be able to be run on one machine or on separate machines.
    - All network communication shall be done with UDP.
- The player controls are limited to :
    - Game <command> where <command> = {START_GAME, END_GAME, JOIN_GAME, RESET_LIFE[3]}
    - Move <direction> where <direction> = {UP, DOWN, LEFT, RIGHT}
    - Deploy <weapon> where <weapon> = {BOMB, DELAYED_BOMB, DETONATE}
    - Player controls are relayed from the player client to the model server as they occur.
    - The server must **concurrently** queue incoming player controls from all clients and process the controls in FIFO order.
        - If a player dies, any subsequent controls in the queue from its player client are ignored. All game logic remains strictly in the model server.
- The view of the game shall be a "graphical" table of the current game state plus that player's status
    - The current game state shall be a string representation of the 2-D floor, plus the remaining number of enemies
    - The player status shall consist of its current power-up (i.e. number of bombs and explosion range).
    - The server shall provide periodic updates of the game state. The period must be configurable - in the third milestone, we will vary the period to see its effects on game latency.
    - The sending of the periodic update must proceed **concurrently** with ongoing processing of player controls. The game state must be **double-buffered**.

[2] Some technical requirements are solely pedagogical – they are put there for teaching reasons and therefore must be met by the students. The word "concurrently" is a clue to such requirements.
[3] RESET_LIFE is a testing hook. According to game rules, a player has only one life. However, during testing, you may want to simply keep playing in order to keep debugging.

- A bomb factory will limit the number of bombs that can be deployed at one time by a player. The bomb scheduler will actually detonate the bombs.
  - The bomb factory shall be implemented as a counting semaphore, to impose the number of bombs that can be deployed.
  - Scheduling of bombs shall use a producer-consumer model – deployed bombs are added to a buffer with their time delay; a scheduler consumes the deployed bombs from the buffer, detonating them after their respective time delay.
  - When a power-up is obtained, a new limit must be assigned for the bomb factory and for the delays.

MILESTONES

The project will have three milestones. Dates will be posted online.

The expectations for each milestone will be posted. The details for each milestone are explained in general terms below, while the exact expectations will be enumerated in the marking scheme posted before the milestone's deadline.

1. During your lab period, each team will demo their current system to their assigned TA. All team members must be present.
2. At the end of your lab period, the team must submit all code and documents for that milestone as a single .zip file. Use the CULearn LAB page for submission.
   - The zip file must include a README.txt that provided detailed instructions for running your program. If the TA cannot run your program, you will lose any marks for parts that cannot be verified by the TA.
3. Select teams will be asked to present aspects of their project on the following Monday class. The other teams will act as peer reviewers.

PROJECT MARKING

The project is worth 25 percent of the final grade. All team members will receive the same grade, if all members contribute equally. See the course outline for the grading of malfunctioning groups. A single peer review will be done at the end of the term to allow confidential concerns about members' contributions. You are encouraged to come forward early – either to your TA or to the instructor – so that the problem can be remedied quickly.