Project Plan

Group Own1:
Joel Lavikainen,
Alvar Martti,
Samuli Mononen,
Noah Nettey

General

Split-screen two players' combat PvP game. Similarity with Worms and Liero. Motivation for this game comes from the legendary Liero game that people tend to play at the guild room of SIK ry. It's a nice game but we thought it's time for us to make an updated clone of it with cool new features. The original graphics aren't also that good so we also wanted to make an even better-looking game.

Scope

General functionality: Players have different weapons that have different effects as well as different moving options. The weapons damage is limited to players only. The game field is static, but it could have some unique features such as invertible gravity etc. We plan to use open source libraries for graphics and game physics. For graphics and input we are using SFML. All the team members have a certain level of familiarity with it already and lots of resources are found for it online. For physics we are using popular Box2D because it was suggested by course staff. It also integrates smoothly with SFML. We will implement sound effects as well as a background music. For sound effects we are going to use an online Flash-tool called Bfxr found from http://www.bfxr.net/**.**

Architecture

The game structure is divided to different classes and functionalities. The main classes are listed below: (More to come possibly)

Main idea: Main game loop that updates the objects by one time step. We use entity component approach, because it simplifies our game loop. This means that all the objects are entities and they are derived from base Entity class. Each entity has their own update, render and logic functions. For checking collisions between say bullets and player, we utilize collision callbacks. Collision callback is a feature in Box2D which eliminates checking for collisions every frame. This is more optimal solution since collisions rarely happen every frame. Objects (eg bullets) are kept in the memory, but dead objects can't be seen or interacted with. For example if there's less than 10 dead bullets in cache and 10 bullets are shot, then new ones need to be constructed. Recycling objects avoid unnecessary processing power lost in construction and destruction of objects.

All graphical objects (players, terrain and projectiles) will have a sprite and a rigid body. Their positions are binded together in the corresponding object class. The rigid body is used to handle the game physics.

Projectile(Abstract class):

- Sprite
- Rigid body
    - Velocity
    - Position
    - Angle
- Status: Alive/Dead
- update, render and logic functions

All different kinds of projectiles inherit the Projectile class.
For example:

Bullet inherits Projectile

- Bullet()
    - set initial position and angle
- update()
    - if(notMoving)
        - apply force to move the rigid body
- logic()
    - no additional logic needed for simple bullet

Weapon(Abstract Class)

- Clip size
- Reload time
- Projectile type
- Shoot function
- variables
    - canShoot

Player:

- Health
- Aim direction
- Kills
- Jetpack status
- Available weapons and their statuses
- Rigid body
- Specific update functions
    - Jump and move etc.
- Moving direction

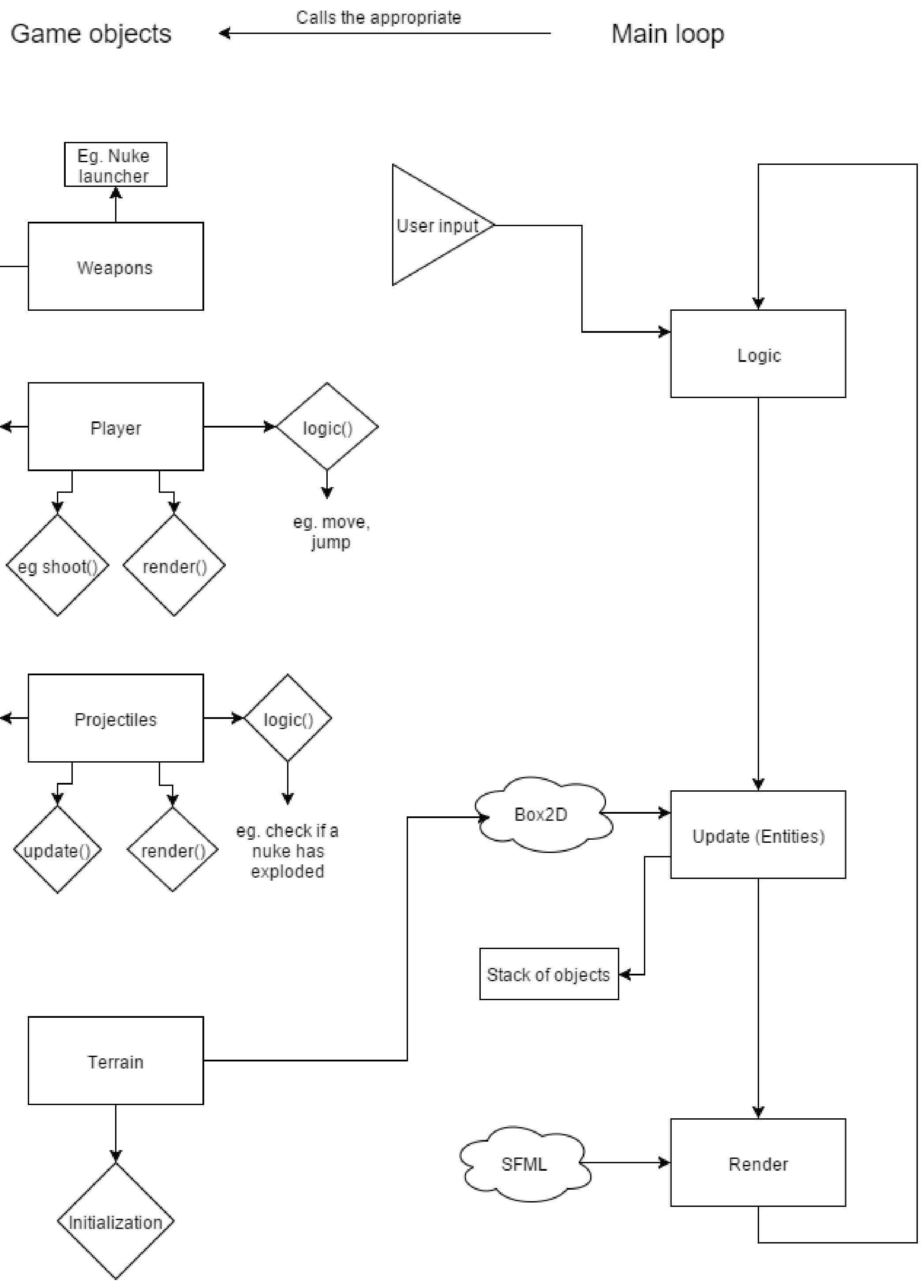
Game field and terrain (abstract design):

- At first hardcoded simple level
- Optionally loading tile levels from files
- Few different textures for solid parts
- Rigid body to interact with other objects

Game physics (abstract design):

- Applies to all entities
- Handles collision detection
- updated from main loop
- We will implement general functions that can control all the game pbjects.

Conceptual design diagram



Optional ideas

- Start menu
- Level loading
- Cool animations
- Tons of weapons (Seeking missiles, player controlled missile, mines)

Preliminary Schedule

Task I: 6. - 10.11.2015

1. Base classes
    a. Players
        i. note: rigid body is the one hardest to implement, might have to be done in task II.
    b. Weapons
    c. Projectiles

Task II: 11. - 18.11.2015

- We know all the base functions which to call

1. Physics engine
    a. Collisions etc.
2. User inputs
3. Associating sprites with rigid bodies
4. Terrain
    a. Rigid body
    b. Static body

Task III: 19. - 23.11.2015

- All fundamental game parts are implemented. Time to put it all together

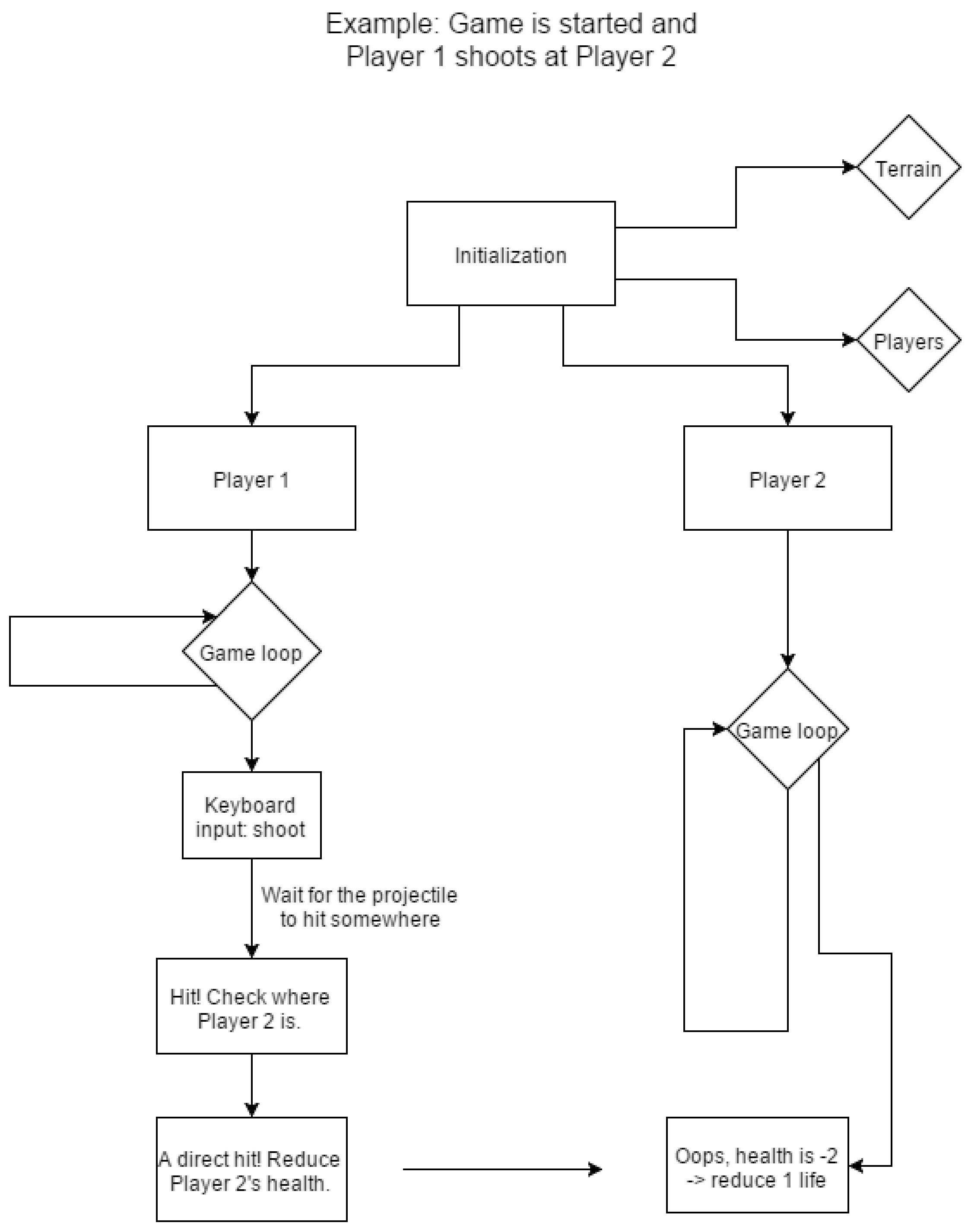
1. Main loop
    a. Calling update and logic functions
    b. Entity stack

Task IV: 23.11. - 7.12.2015

- Time to make it look like a game

1. Sounds
2. Camera following
3. GUI
4. Splitscreen
5. Etc.

Gameplay


Example: Game is started and Player 1 shoots at Player 2
Initialization
Terrain
Players
Player 1
Player 2
Game loop
Game loop
Keyboard input: shoot
Wait for the projectile to hit somewhere
Hit! Check where Player 2 is.
A direct hit! Reduce Player 2's health.
Oops, health is -2 -> reduce 1 life