Class and Package Diagram for System TwitterNetHack

Assignment in the course - PA1435 Objektorienterad design

17/5 - 2017

Name	Social S. #	Thinking	Writing
Patrik Sjölund	19960604-9493	20%	20%
Tristan Smith	19960622-9574	20%	20%
Jesper Åkesson	19960821-0275	20%	20%
Gustaf Fagerström	19960807-6692	20%	20%
Joakim Bresche	19940914-6538	20%	20%

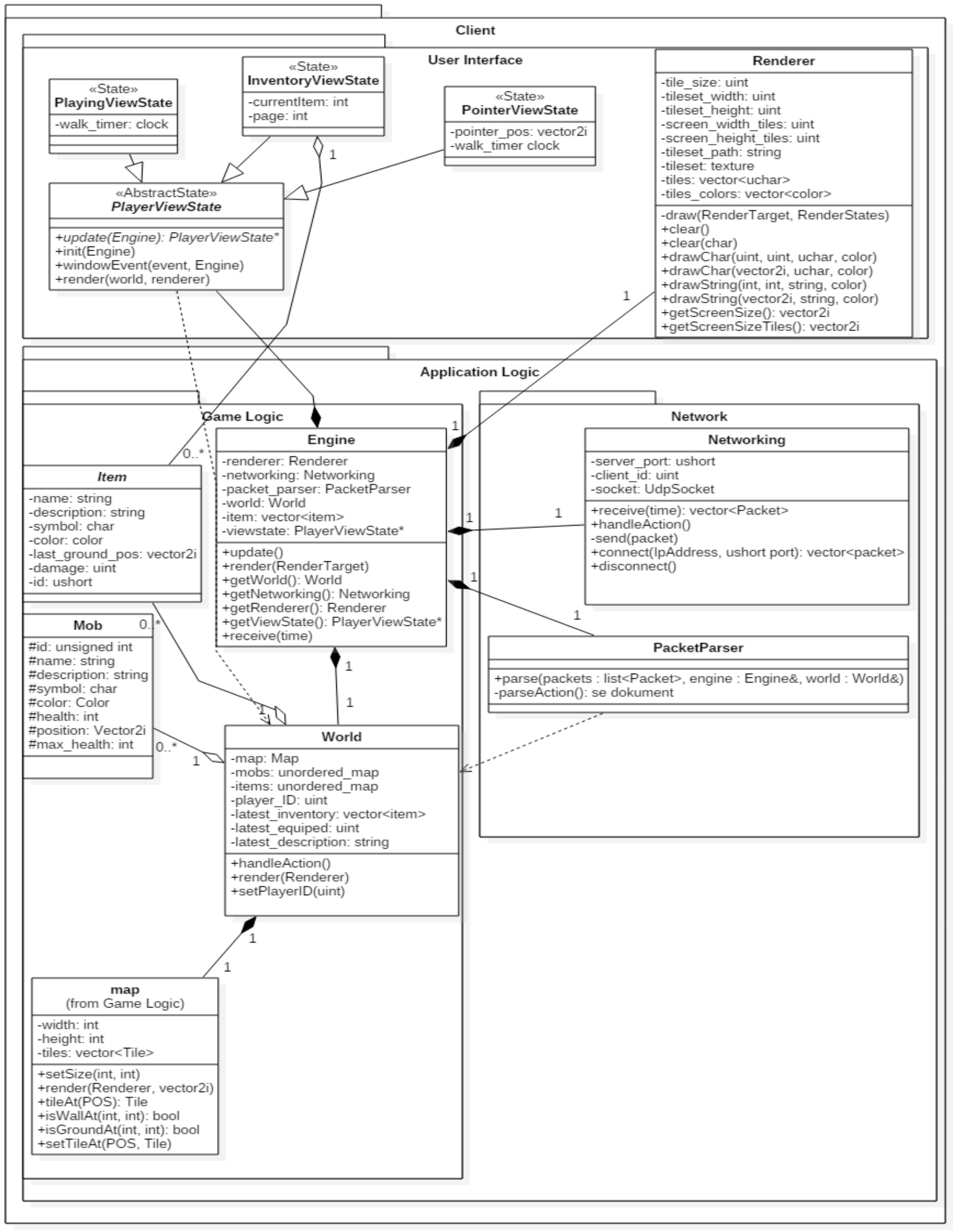
Link to implementation:
https://github.com/Trisslotten/OOD-PA1435-Tristan-Smith-Implementation

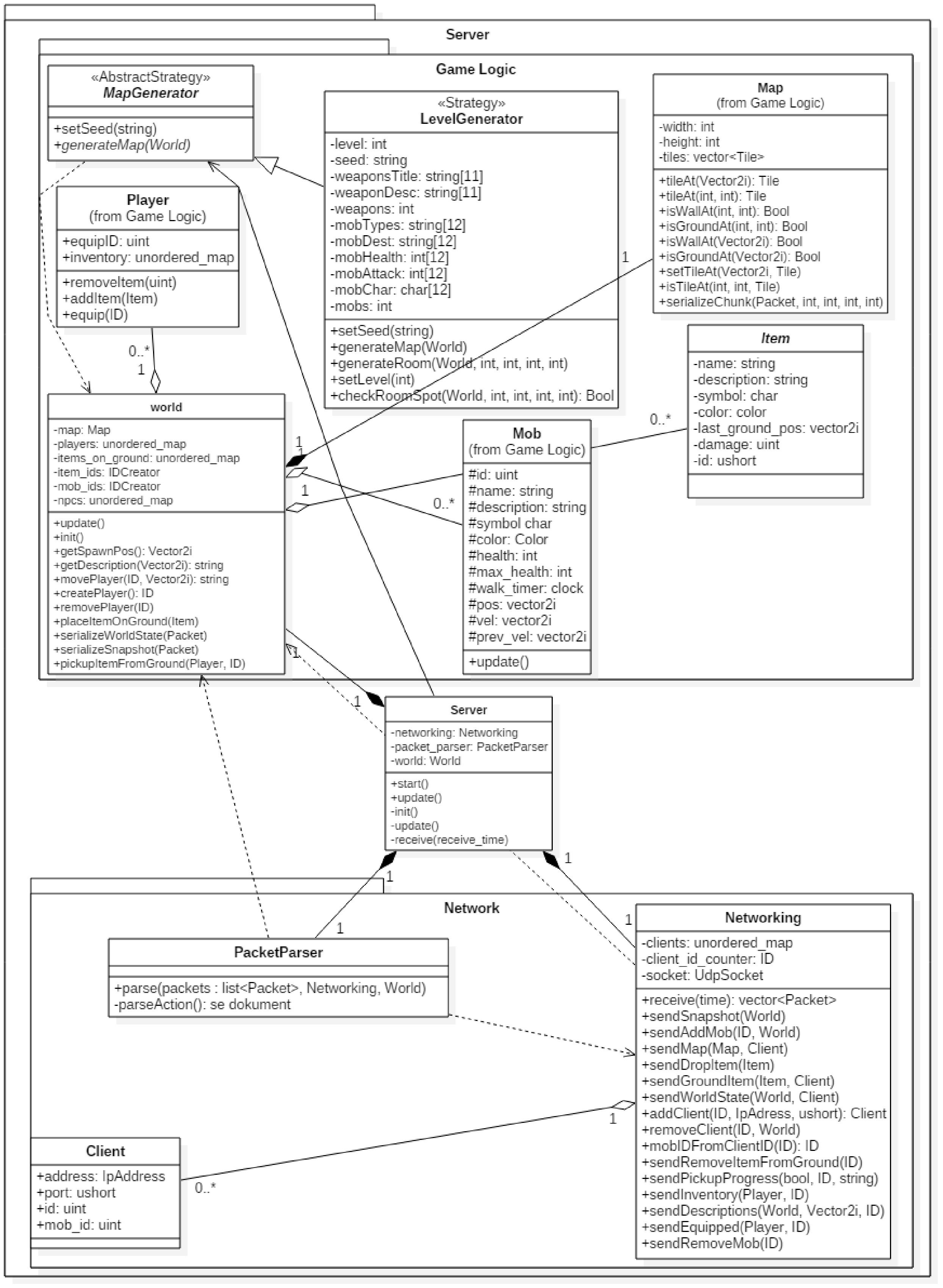
System description

The system is a clone of the game “nethack”, a game where the player(s) try to defeat a so called “dungeon” full of monsters and traps. Starting out on the first level, you continue on downwards for which each level becomes progressively harder. In this system the levels continue forever until the player eventually loses unlike the original where the game actually has an ending.

This clone is going to include multiplayer and random-generated dungeons where they are either generated by the twitter api or a pseudo random generator, thus the the clone is not going to be turn-based as the original game, however it is still going to be grid based as the original.

Class and Package Diagram




Server
Game Logic
«AbstractStrategy»
MapGenerator
+setSeed(string)
+generateMap(World)
«Strategy»
LevelGenerator
-level: int
-seed: string
-weaponsTitle: string[11]
-weaponDesc: string[11]
-weapons: int
-mobTypes: string[12]
-mobDest: string[12]
-mobHealth: int[12]
-mobAttack: int[12]
-mobChar: char[12]
-mobs: int
+setSeed(string)
+generateMap(World)
+generateRoom(World, int, int, int, int)
+setLevel(int)
+checkRoomSpot(World, int, int, int, int): Bool
Map
(from Game Logic)
-width: int
-height: int
-tiles: vector<Tile>
+tileAt(Vector2i): Tile
+tileAt(int, int): Tile
+isWallAt(int, int): Bool
+isGroundAt(int, int): Bool
+isWallAt(Vector2i): Bool
+isGroundAt(Vector2i): Bool
+setTileAt(Vector2i, Tile)
+isTileAt(int, int, Tile)
+serializeChunk(Packet, int, int, int, int)
Player
(from Game Logic)
+equipID: uint
+inventory: unordered_map
+removeItem(uint)
+addItem(Item)
+equip(ID)
0..*
1
1
Item
-name: string
-description: string
-symbol: char
-color: color
-last_ground_pos: vector2i
-damage: uint
-id: ushort
0..*
world
-map: Map
-players: unordered_map
-items_on_ground: unordered_map
-item_ids: IDCreator
-mob_ids: IDCreator
-npcs: unordered_map
+update()
+init()
+getSpawnPos(): Vector2i
+getDescription(Vector2i): string
+movePlayer(ID, Vector2i): string
+createPlayer(): ID
+removePlayer(ID)
+placeItemOnGround(Item)
+serializeWorldState(Packet)
+serializeSnapshot(Packet)
+pickupItemFromGround(Player, ID)
1
1
1
0..*
Mob
(from Game Logic)
#id: uint
#name: string
#description: string
#symbol char
#color: Color
#health: int
#max_health: int
#walk_timer: clock
#pos: vector2i
#vel: vector2i
#prev_vel: vector2i
+update()
1
1
Server
-networking: Networking
-packet_parser: PacketParser
-world: World
+start()
+update()
-init()
-update()
-receive(receive_time)
1
1
1
1
Network
PacketParser
+parse(packets : list<Packet>, Networking, World)
-parseAction(): se dokument
Networking
-clients: unordered_map
-client_id_counter: ID
-socket: UdpSocket
+receive(time): vector<Packet>
+sendSnapshot(World)
+sendAddMob(ID, World)
+sendMap(Map, Client)
+sendDropItem(Item)
+sendGroundItem(Item, Client)
+sendWorldState(World, Client)
+addClient(ID, IpAdress, ushort): Client
+removeClient(ID, World)
+mobIDFromClientID(ID): ID
+sendRemoveItemFromGround(ID)
+sendPickupProgress(bool, ID, string)
+sendInventory(Player, ID)
+sendDescriptions(World, Vector2i, ID)
+sendEquipped(Player, ID)
+sendRemoveMob(ID)
Client
+address: IpAddress
+port: ushort
+id: uint
+mob_id: uint
1
0..*

Client.Network - Networking
handleAction() represents the following functions:

```
void testMove(sf::Vector2i vel);
void sendPickup();
void sendRequestInventory();
void sendRequestDescriptions(sf::Vector2i pos);
void sendDropItem(ID item_id);
void sendEquipItem(ID item_id);
void sendRequestEquipped();
```

Client.Network - PacketParser
parseAction() represents the following functions:

```
void debugMove(sf::Packet & packet, Engine & engine);
void confirmJoin(sf::Packet & packet, Engine & engine);
void addMob(sf::Packet packet, Engine& engine);
void removeMob(sf::Packet packet, Engine& engine);
void parseWorldState(sf::Packet packet, Engine& engine);
void dropItem(sf::Packet packet, Engine& engine);
void removeItem(sf::Packet packet, Engine& engine);
void parseMapSize(sf::Packet packet, Engine& engine);
void parseMapChunk(sf::Packet packet, Engine& engine);
void serverShutdown(Engine& engine);
void pickupSuccess(sf::Packet packet, Engine& engine);
void pickupFailed(sf::Packet packet, Engine& engine);
void receiveInventory(sf::Packet packet, Engine& engine);
void parseDescriptions(sf::Packet packet, Engine& engine);
void receiveEquipped(sf::Packet packet, Engine& engine);
```

Server.Network - PacketParser
parseAction() represents the following functions:

```
void joinServer(Packet& packet, Networking& networking, World& world);
void disconnectClient(Packet& packet, Networking& networking, World& world);
void dropItem(Packet& packet, Networking& networking, World& world);
void parseMovePlayer(Packet& packet, Networking& networking, World& world);
void pickupItem(Packet& packet, Networking& networking, World& world);
void requestInventory(Packet& packet, Networking& networking, World& world);
void requestDescriptions(Packet& packet, Networking& networking, World& world);
void equipItem(Packet& packet, Networking& networking, World& world);
void requestEquipped(Packet& packet, Networking& networking, World& world);
```

Package descriptions (Client):

1. **User Interface**

All classes used to display the menus and world to the player, as well as classes used for player interaction.

2. **Application logic**

All backend updating and handling of entities.

2.1 Game logic
All game objects.

2.2 Networking
All client-side networking. Incoming packets are parsed and sent along to different application logic functions.

Package descriptions (Server):

1. **Game logic**

Handling of all objects and interactions in the world.

2. **Network**

All server-side networking. Incoming packets are parsed in the network-side of the server and passed along to different game logic event-functions.