KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

OBJEKTINIS PROGRAMŲ PROJEKTAVIMAS (T120B516)

Laboratorinis darbas Nr. 1

Atliko:
Paulius Ratkevičius IFF 8/12
Dovydas Bražas IFF-8/6
Antanas Stepanauskas IFF-8/6

Priėmė:
Andrej Ušaniov

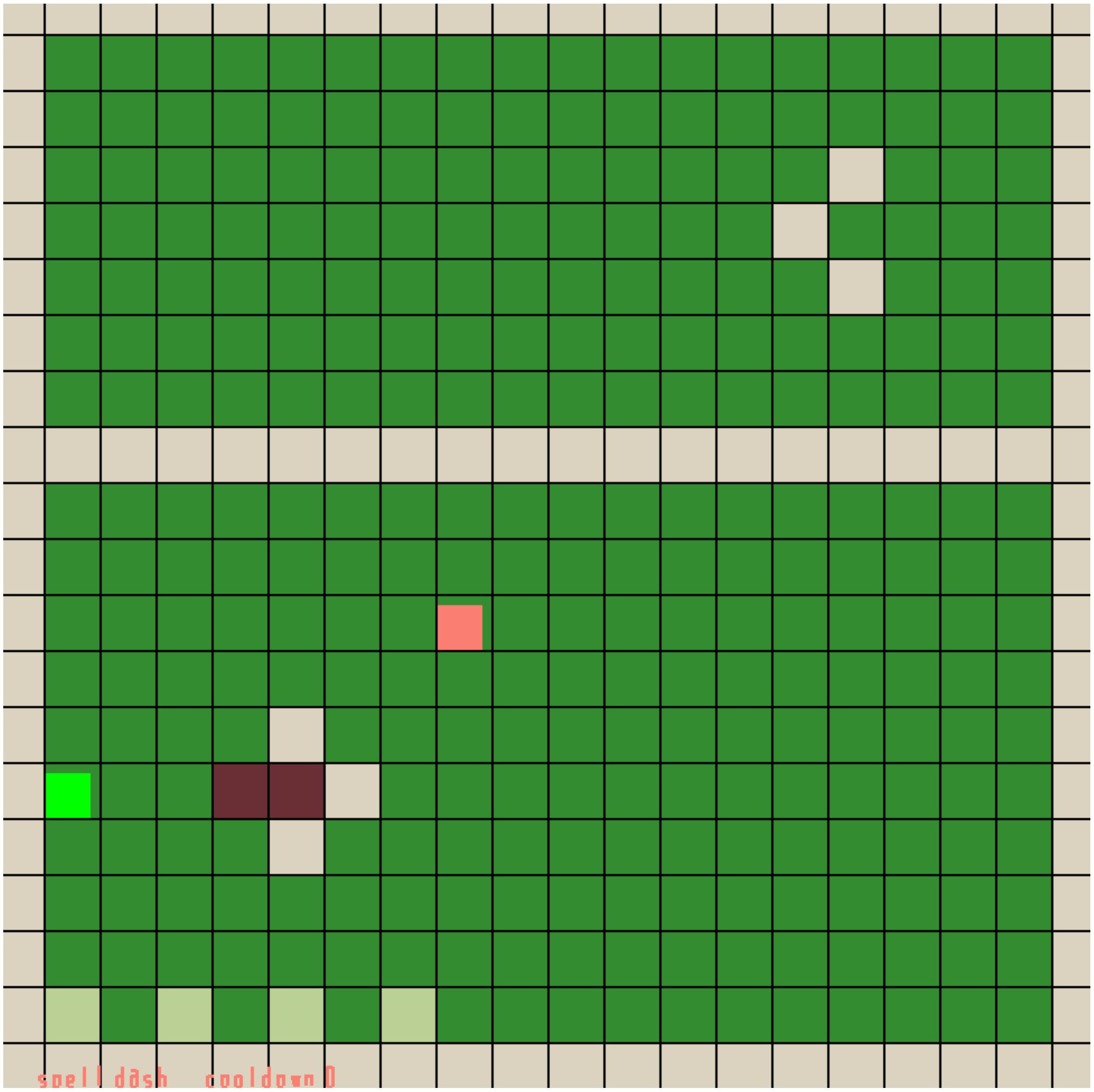
KAUNAS, 2021

TURINYS

Contents

1. PROJEKTO APRAŠYMAS

Žaidimas, kuriame strategiškai dėlioji bombas, kuriomis gali sunaikinti kliūtis bei priešus, išvengi-nėji spąstų bei naudoji įgytas galias kad įgautum pranašumą prieš savo varžovą.



1.1 pav. Žaidimo prototipo nuotrauka

2. ŽAIDIMO REIKALAVIMAI

2.1. ŽAIDIMO LYGIAI

Žaidimas susidės iš trijų lygių, kurie vienas nuo kito skirsis savo žaidimo strategijomis bei sudėtingumu.

2.2. PIRMASIS LYGIS

Jame bus sukurtas pasaulis, kuris susidės iš sienų, kurių kiaurai pereiti negalima, bet galima jas susprogdinti naudojant bombas, bei žemės, per kurią žaidėjas gales laisvai vaikščioti. Žaidimo tiklsas nugalėti savo priešininką, taktiškai naikinant sienas. Žaidėjai turės po 3 gyvybes, viena gyvybė yra prarandama jeigu savo arba priešininko bomba sprogsta šalia.

2.3. ANTRAS LYGIS

Antrame lygyje atsiranda nauja kliūtis- tai spąstai, kurie sugeneruojami sukuriant 2 lygio pasaulį. Visi spąstai atrodo taip pat tik žaidėjai nežino, ką jis gali padaryti, tik žino tą kad visi spąstai jį užsaldys. Taip pat spąstai gali žaidėja sulėtinti, nuimti gyvybę ar jį kažkur nukelti.

2.4. TREČIAS LYGIS

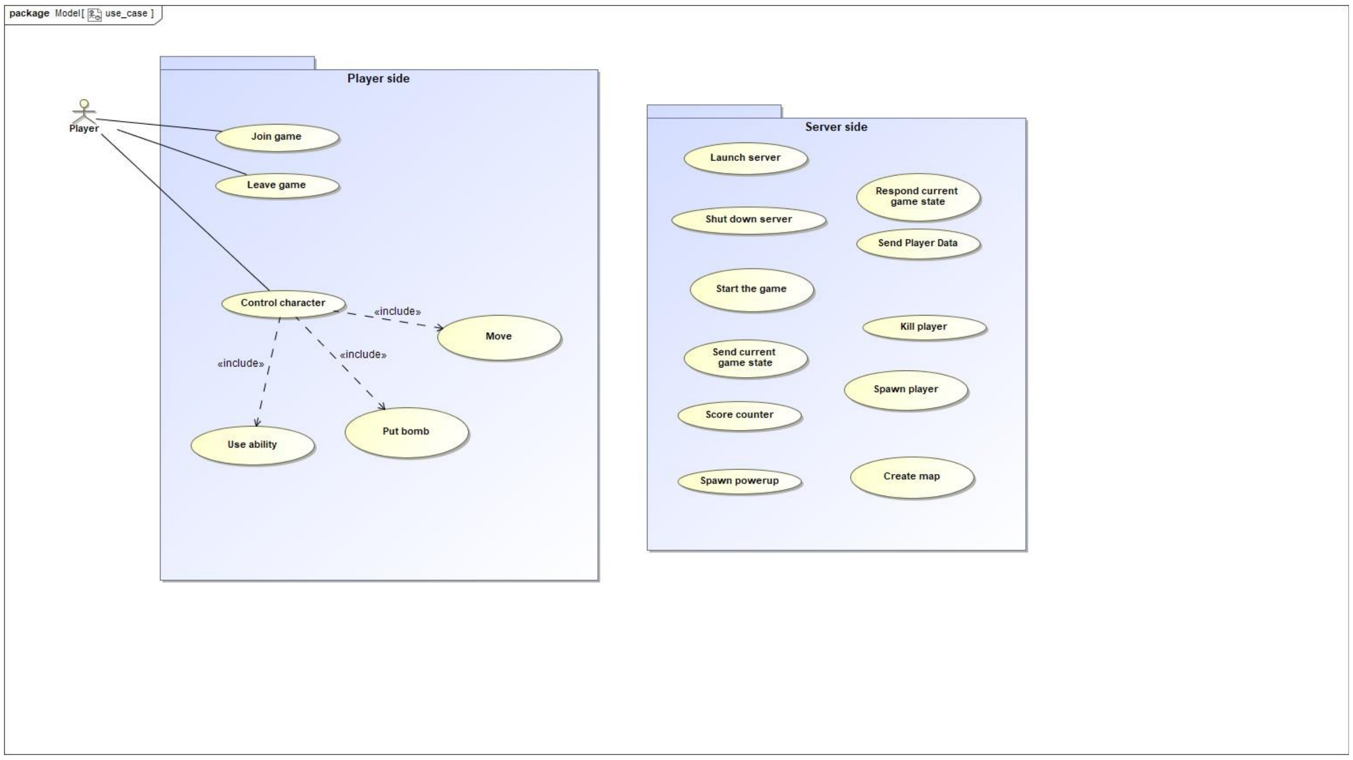
Trečiame lygyje atsiranda žaidėjo įgudis. Kai 3 lygio pasaulis yra sugeneruotas kas 10 sekundžių atsiranda žaidėjo įgudis, ant kurio užlipus žaidėjas gauna viena iš 4 pagerinimų:

- Sulėtinimas priešininko. Galima naudoti kas 30 sekundžių.
- Nusikelimas į naują vietą. Galima naudoti kas 20 sekundžių.
- Sienų peršokimas. Galima naudoti kas 15 sekundžių.
- Greitas žaidėjo paslinkimas. Galima naudoti kas 15 sekundžių.

3. PROJEKTUI NAUDOTOS TECHNOLOGIJOS

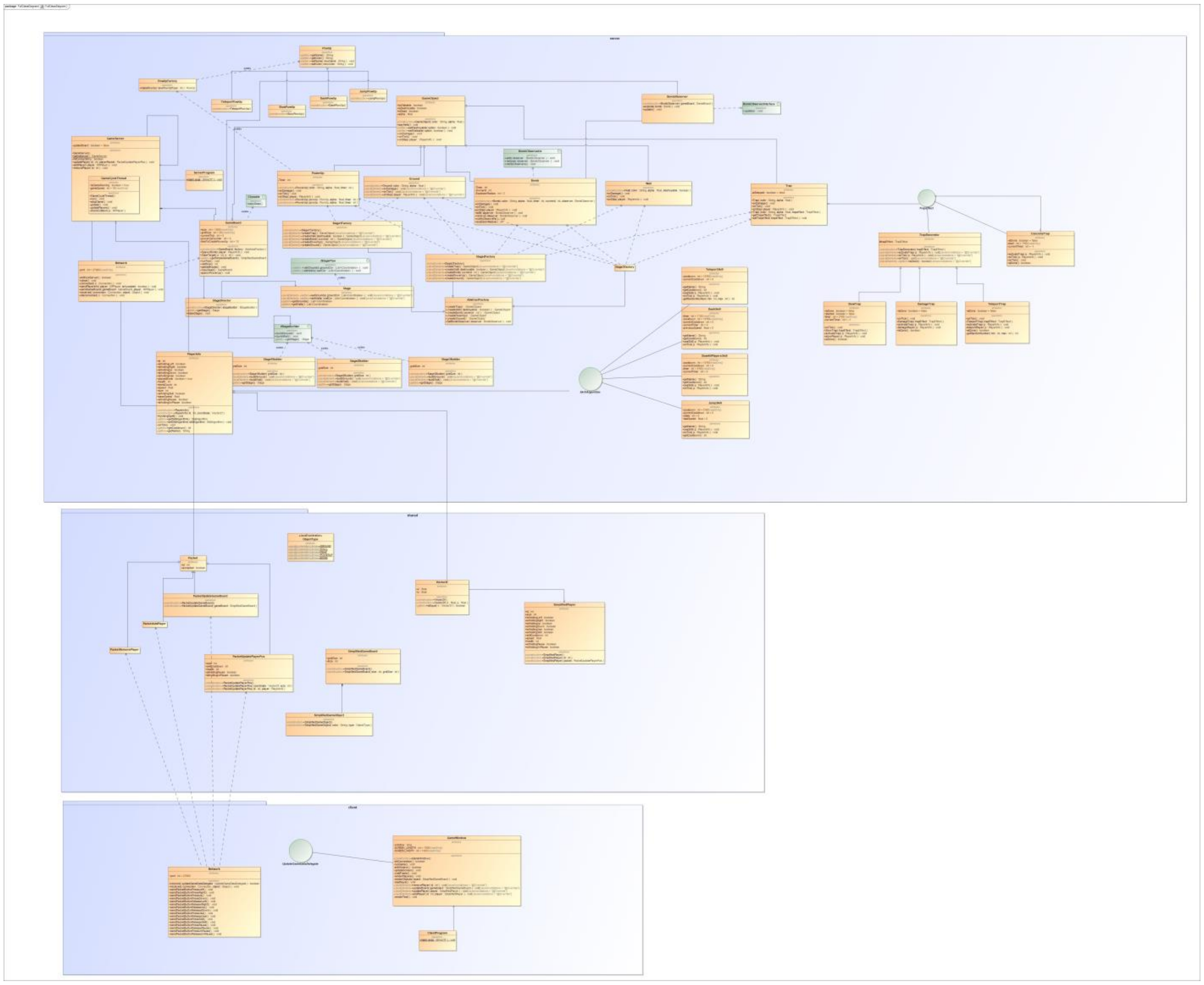
Serverio ir kliento komunikacijai naudotas Kryonet. Programos lango sukūrimui ir piešimui buvo pasitelkta lwjgl ir OpenGL 1.1 įrankiai.

4. USE CASE DIAGRAMA



4.2 pav. Use case diagrama

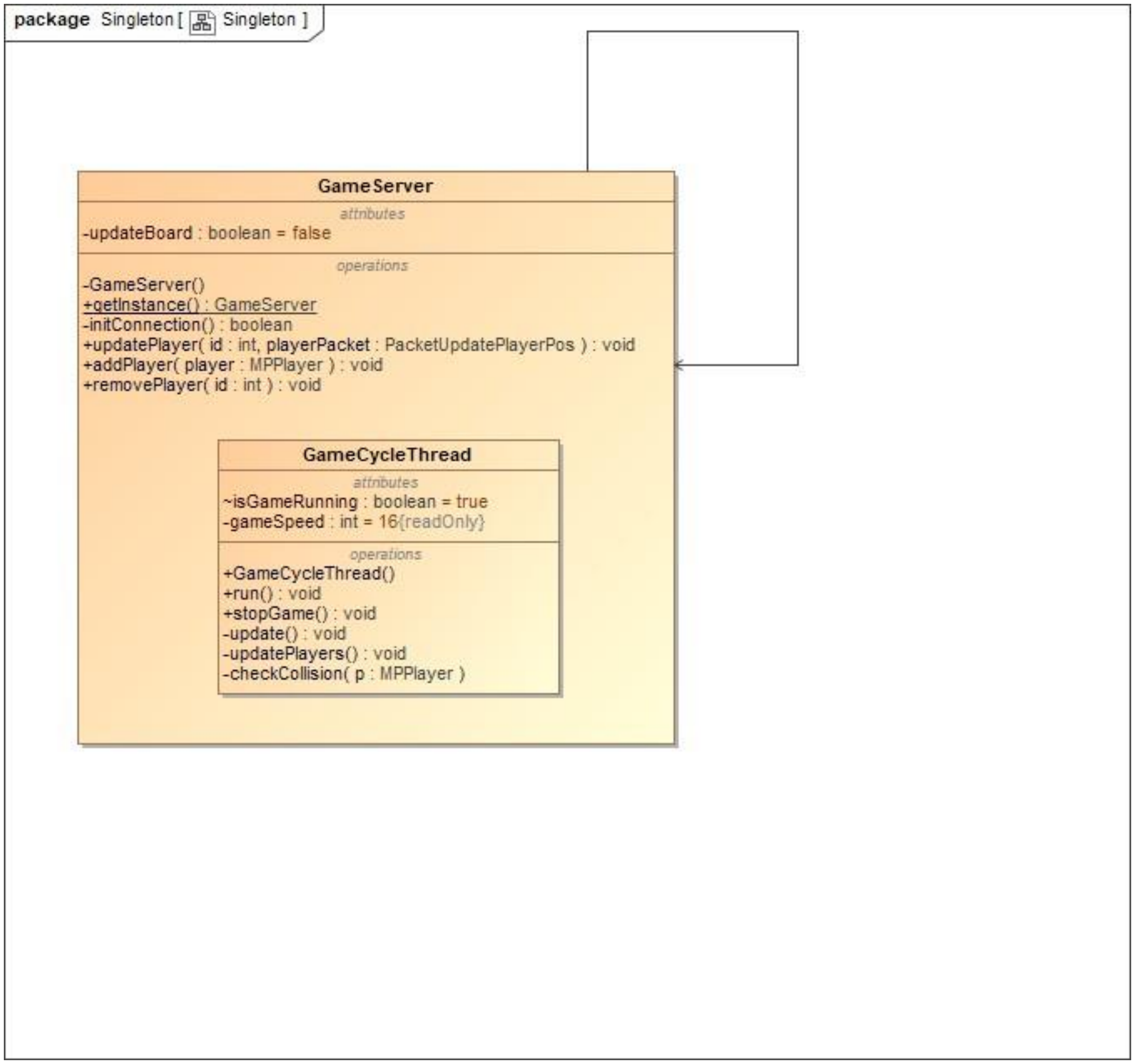
5. KLASIŲ DIAGRAMA



5.3 pav. Klasių diagrama

5.1. ŠABLONAI

5.1.1. Singleton



package Singleton [Singleton]
GameServer
attributes
-updateBoard : boolean = false
operations
-GameServer()
+getInstance() : GameServer
-initConnection() : boolean
+updatePlayer(id : int, playerPacket : PacketUpdatePlayerPos) : void
+addPlayer(player : MPPlayer) : void
+removePlayer(id : int) : void
GameCycleThread
attributes
~isGameRunning : boolean = true
-gameSpeed : int = 16{readOnly}
operations
+GameCycleThread()
+run() : void
+stopGame() : void
-update() : void
-updatePlayers() : void
-checkCollision(p : MPPlayer)

5.4 pav. Singleton diagrama

Šis algoritmas buvo pasirinktas dėl to, jog vienu metu gali vykti tik vienas žaidimas su bendra logika. Taip pat šią žaidimo klasę paranku pasiekti visame projekte.

```java
package server;

import java.util.HashMap;
import java.util.Map;

import shared.Vector2f;
import shared.PacketUpdatePlayerPos;

class GameServer
{
  private static GameServer gameServer = null;

  protected volatile GameBoard gameBoard;
  protected volatile Map<Integer, MPPlayer> players;
  protected volatile Network network;
  private GameCycleThread thread;
  private Stage1Factory stage1factory;
  private boolean updateBoard = false;

  private GameServer()
  {
    //Init Connection
    if (!initConnection())
    {
      System.err.println("ERROR Connecting to host");
      return;
    }

    this.players = new HashMap<Integer, MPPlayer>();
    this.stage1factory = new Stage1Factory();
    this.gameBoard = new GameBoard(stage1factory);

    this.thread = new GameCycleThread();
    this.thread.start();
}

public static GameServer getInstance()
{
    if (gameServer == null)
    {
        gameServer = new GameServer();
    }
    return gameServer;
}

private boolean initConnection()
{
    this.network = new Network();

    if (!this.network.initKryoServer())
    {
        return false;
    }

    return true;
}

public void updatePlayer(int id, PacketUpdatePlayerPos playerPacket)
```

```
{
  MPPlayer player = players.get(id);
  if (player != null)
  {
    player.isHoldingUp = playerPacket.isHoldingUp != null ?
 playerPacket.isHoldingUp : player.isHoldingUp;
    player.isHoldingDown = playerPacket.isHoldingDown != null ?
 playerPacket.isHoldingDown : player.isHoldingDown;
    player.isHoldingLeft = playerPacket.isHoldingLeft != null ?
 playerPacket.isHoldingLeft : player.isHoldingLeft;
    player.isHoldingRight = playerPacket.isHoldingRight != null ?
 playerPacket.isHoldingRight : player.isHoldingRight;
    player.isHoldingUse = playerPacket.isHoldingUse != null ?
 playerPacket.isHoldingUse : player.isHoldingUse;
    player.isHoldingSkill = playerPacket.isHoldingSkill != null ?
 playerPacket.isHoldingSkill : player.isHoldingSkill;
    players.put(player.id, player);
  }
}

public void addPlayer(MPPlayer player)
{
  this.players.put(player.c.getID(), player);
  this.network.sendGameBoard(gameBoard, player);

}

public void removePlayer(int id)
{

    players.remove(id);
}


    private class GameCycleThread extends Thread
    {
        volatile boolean isGameRunning = true;
        private final int gameSpeed = 16; //The lower the number the
  faster the game is

        public GameCycleThread()
        {
            this.isGameRunning = true;
        }

            public void run()
            {
                while (this.isGameRunning)
                {
                    try
                    {
                        //Probably should use Timer instead
                        Thread.sleep(gameSpeed);
                    this.update();
                }
                    catch (InterruptedException e)
                    {
                    e.printStackTrace();
            this.stopGame();
          }
        }
    }

    public void stopGame()
    {
      this.isGameRunning = false;

      //network should probably be closed by the parent
      network.close();
    }

    private void update()
    {
      updatePlayers();
      gameBoard.runTick();
    }

    private void updatePlayers()
    {
    for(MPPlayer p : players.values())
    {
      if (p.isHoldingPause)
    {

    }

      if (p.isHoldingSkill)
```

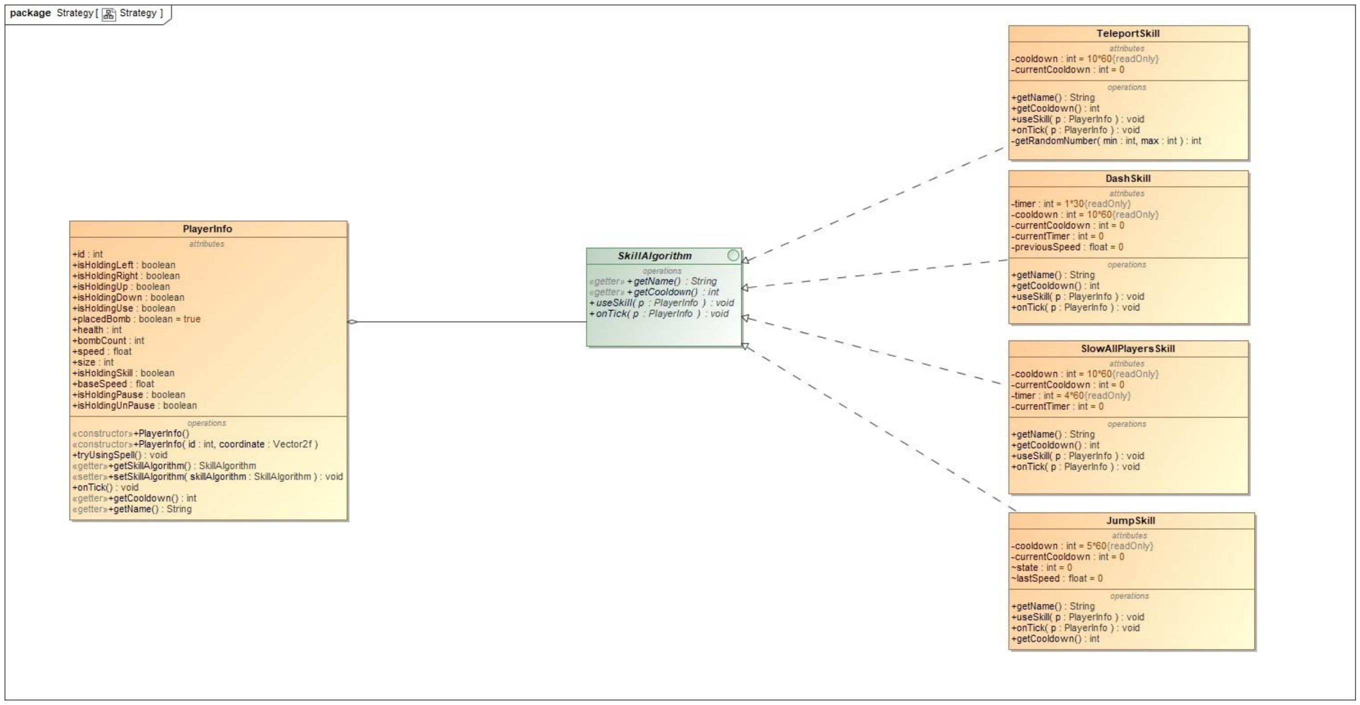
```
          {
             p.tryUsingSpell();
          }

          p.onTick();

       if (p.isHoldingUse)
       {
          gameBoard.SpawnBomb(p);
       }

       p.coordinate = checkCollision(p);

       network.sendGameBoard(gameBoard, p);
          network.sendPlayerInfo(p, true);
       }
       }

       private Vector2f checkCollision(MPPlayer p)
       {

          Vector2f coordsAfterMove = new Vector2f(p.coordinate.x, p.
coordinate.y);

          float padding = 0.001f;
    float cellSize = gameBoard.cellSize();

    boolean moveX = true;
    boolean moveY = true;

    if (p.isHoldingLeft)
    {
      coordsAfterMove.x -= p.speed;
    }

    if (p.isHoldingRight)
    {

      coordsAfterMove.x += p.speed;
    }

    boolean collidingLeft = ((int)coordsAfterMove.x / cellSize -
  padding)<((int)p.coordinate.x / cellSize);
    boolean collidingRight=(((int)coordsAfterMove.x+p.size+
  padding)/cellSize)>(((int)p.coordinate.x+p.size)/cellSize);
    boolean isCollidingX = collidingLeft || collidingRight;
    moveX = !(coordsAfterMove.x <= 0 || coordsAfterMove.x >=
  gameBoard.size - p.size);
    //Some smoothing when going around edges would be nice
    if (isCollidingX && moveX)
    {
      int x = 0, y = 0, y1 = 0;
      if (collidingLeft)
      {
        x = (int) ((coordsAfterMove.x) / cellSize);
        y = (int) (p.coordinate.y / cellSize);
        y1 = (int) ((p.coordinate.y + p.size) / cellSize);

```

```
            }
            if (collidingRight)
            {
                x = (int) ((coordsAfterMove.x + p.size) / cellSize);
                y = (int) (p.coordinate.y / cellSize);
                y1 = (int) ((p.coordinate.y + p.size) / cellSize);

            }

            if (y == y1)
            {
                moveX = gameBoard.objects[x][y].isWalkable;
                gameBoard.objects[x][y].onStep(p);
            }
            else
            {
                moveX = gameBoard.objects[x][y].isWalkable && gameBoard.
    objects[x][y1].isWalkable;
                gameBoard.objects[x][y].onStep(p);
                gameBoard.objects[x][y1].onStep(p);
            }
        }


        if (p.isHoldingUp)
        {
            coordsAfterMove.y += p.speed;
        }

```

```
      if (p.isHoldingDown)
      {
        coordsAfterMove.y -= p.speed;
      }

      boolean collidingUp = (((int)coordsAfterMove.y + p.size - padding
  ) / cellSize) > (((int)p.coordinate.y + p.size) / cellSize);
      boolean collidingDown = ((int)coordsAfterMove.y/cellSize+
  padding)< ((int)p.coordinate.y / cellSize);
      boolean isCollidingY = collidingUp || collidingDown;
      moveY = !(coordsAfterMove.y <= 0 || coordsAfterMove.y >=
  gameBoard.size - p.size);

      //Some smoothing when going around edges would be nice
      if (isCollidingY && moveY)
      {
        int x = 0, x1 = 0, y = 0;

        if (collidingUp)
        {
          y = (int) ((coordsAfterMove.y + p.size) / cellSize);
          x = (int) (p.coordinate.x / cellSize);
          x1 = (int) ((p.coordinate.x + p.size) / cellSize);

        }
        if (collidingDown)
        {
          y = (int) ((coordsAfterMove.y) / cellSize);
          x = (int) (p.coordinate.x / cellSize);
```

```
                x1 = (int) ((p.coordinate.x + p.size) / cellSize);
            }

            if (x == x1)
            {
                moveY = gameBoard.objects[x][y].isWalkable;
                gameBoard.objects[x][y].onStep(p);
            }
            else
            {
                moveY = gameBoard.objects[x][y].isWalkable && gameBoard.
    objects[x1][y].isWalkable;
                gameBoard.objects[x][y].onStep(p);
                gameBoard.objects[x1][y].onStep(p);
            }
        }

        coordsAfterMove.x = moveX ? coordsAfterMove.x : p.coordinate.x;
        coordsAfterMove.y = moveY ? coordsAfterMove.y : p.coordinate.y;

        return coordsAfterMove;

            }

      }

}
```

5.1.2. Strategy


package Strategy [Strategy]
PlayerInfo
attributes
+id : int
+isHoldingLeft : boolean
+isHoldingRight : boolean
+isHoldingUp : boolean
+isHoldingDown : boolean
+isHoldingUse : boolean
+placedBomb : boolean = true
+health : int
+bombCount : int
+speed : float
+size : int
+isHoldingSkill : boolean
+baseSpeed : float
+isHoldingPause : boolean
+isHoldingUnPause : boolean
operations
«constructor»+PlayerInfo()
«constructor»+PlayerInfo(id : int, coordinate : Vector2f)
+tryUsingSpell() : void
«getter»+getSkillAlgorithm() : SkillAlgorithm
«setter»+setSkillAlgorithm(skillAlgorithm : SkillAlgorithm) : void
+onTick() : void
«getter»+getCooldown() : int
«getter»+getName() : String
SkillAlgorithm
operations
«getter» +getName() : String
«getter» +getCooldown() : int
+useSkill(p : PlayerInfo) : void
+onTick(p : PlayerInfo) : void
TeleportSkill
attributes
-cooldown : int = 10*60{readOnly}
-currentCooldown : int = 0
operations
+getName() : String
+getCooldown() : int
+useSkill(p : PlayerInfo) : void
+onTick(p : PlayerInfo) : void
-getRandomNumber(min : int, max : int) : int
DashSkill
attributes
-timer : int = 1*30{readOnly}
-cooldown : int = 10*60{readOnly}
-currentCooldown : int = 0
-currentTimer : int = 0
-previousSpeed : float = 0
operations
+getName() : String
+getCooldown() : int
+useSkill(p : PlayerInfo) : void
+onTick(p : PlayerInfo) : void
SlowAllPlayersSkill
attributes
-cooldown : int = 10*60{readOnly}
-currentCooldown : int = 0
-timer : int = 4*60{readOnly}
-currentTimer : int = 0
operations
+getName() : String
+getCooldown() : int
+useSkill(p : PlayerInfo) : void
+onTick(p : PlayerInfo) : void
JumpSkill
attributes
-cooldown : int = 5*60{readOnly}
-currentCooldown : int = 0
~state : int = 0
~lastSpeed : float = 0
operations
+getName() : String
+useSkill(p : PlayerInfo) : void
+onTick(p : PlayerInfo) : void
+getCooldown() : int

5.5 pav. Strategy diagrama

Strategy šablono poreikis atsirado norint įterpti skirtinga funkcionalumą žaidėjui nepakeičiant kaip jis jį naudoja. Pasitelkti skirtingi algoritmai skirtingiems įgudžiams.

```java
package server;

public interface SkillAlgorithm
{
    int getCooldown();
    void useSkill(PlayerInfo p);
    void onTick(PlayerInfo p);
    String getName();
}
```

```java
package server;

```

```java
public class JumpSkill implements SkillAlgorithm
{
  private final int cooldown = 5 * 60; //5 seconds
  private final String name = "Jump";
  private int currentCooldown = 0;
  int state = 0;
  float lastSpeed = 0;
  @Override
  public void useSkill(PlayerInfo p)
  {
    if (this.currentCooldown == 0)
    {
      this.state = 2;
      this.currentCooldown = this.cooldown;
    }
  }

  @Override
  public void onTick(PlayerInfo p)
  {
    if (this.currentCooldown > 0)
    {
      this.currentCooldown --;
    }

    if (this.state == 2)
    {
      GameServer gameserver = GameServer.getInstance();
      this.lastSpeed = p.speed;
```

```java
            p.speed = (gameserver.gameBoard.size / gameserver.gameBoard.
    gridSize)*2;
            this.state = 1;
        }
        else if (this.state == 1)
        {
            p.speed = lastSpeed;
            this.state=0;
        }
    }

    @Override
    public String getName()
    {
        return this.name;
    }

    @Override
    public int getCooldown()
    {
        return this.currentCooldown;
    }
}
```

```java
package server;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import shared.SimplifiedPlayer;

public class SlowAllPlayersSkill implements SkillAlgorithm
{
  private final int cooldown = 10 * 60; //10 seconds
  private final String name = "Slow";
  private int currentCooldown = 0;
  private final int timer = 4 * 60; //4 seconds
  private int currentTimer = 0;
  protected volatile Map<Integer, SimplifiedPlayer> simplifiedPlayers;
  @Override
  public void useSkill(PlayerInfo p)
  {
    if (this.currentCooldown == 0)
    {
      GameServer gameserver = GameServer.getInstance();
      simplifiedPlayers = new HashMap<Integer, SimplifiedPlayer>();
        for(MPPlayer singlePlayer : gameserver.players.values())
        {
        if (singlePlayer != null)
        {
          if (singlePlayer != p)
          {
            SimplifiedPlayer player = new SimplifiedPlayer();
            player.id = p.id;
            player.speed = p.speed;
            float newSpeed = p.speed * 0.5f;
```

```
                singlePlayer.speed = newSpeed;
                this.simplifiedPlayers.put(player.id, player);
            }

        }

            }
        this.currentTimer = timer;
        this.currentCooldown = this.cooldown;
    }
}

@Override
public void onTick(PlayerInfo p)
{
    if (this.currentCooldown > 0)
    {
        this.currentCooldown --;
    }
    if (this.currentTimer > 1)
    {
        this.currentTimer--;
    }
    else if (this.currentTimer == 1)
    {
        GameServer gameserver = GameServer.getInstance();
            for(MPPlayer singlePlayer : gameserver.players.values())
            {
            if (singlePlayer != null)
            {
               if (singlePlayer != p)
               {
                  SimplifiedPlayer simplifiedPlayer = this.simplifiedPlayers.
   get(singlePlayer.id);
                  if (simplifiedPlayer != null)
                  {
                     singlePlayer.speed = simplifiedPlayer.speed;
                  }
               }

            }

               }
         this.currentTimer --;
      }

   }

   @Override
   public String getName()
   {
      return this.name;
   }

   @Override
   public int getCooldown()
   {
      return this.currentCooldown;
```

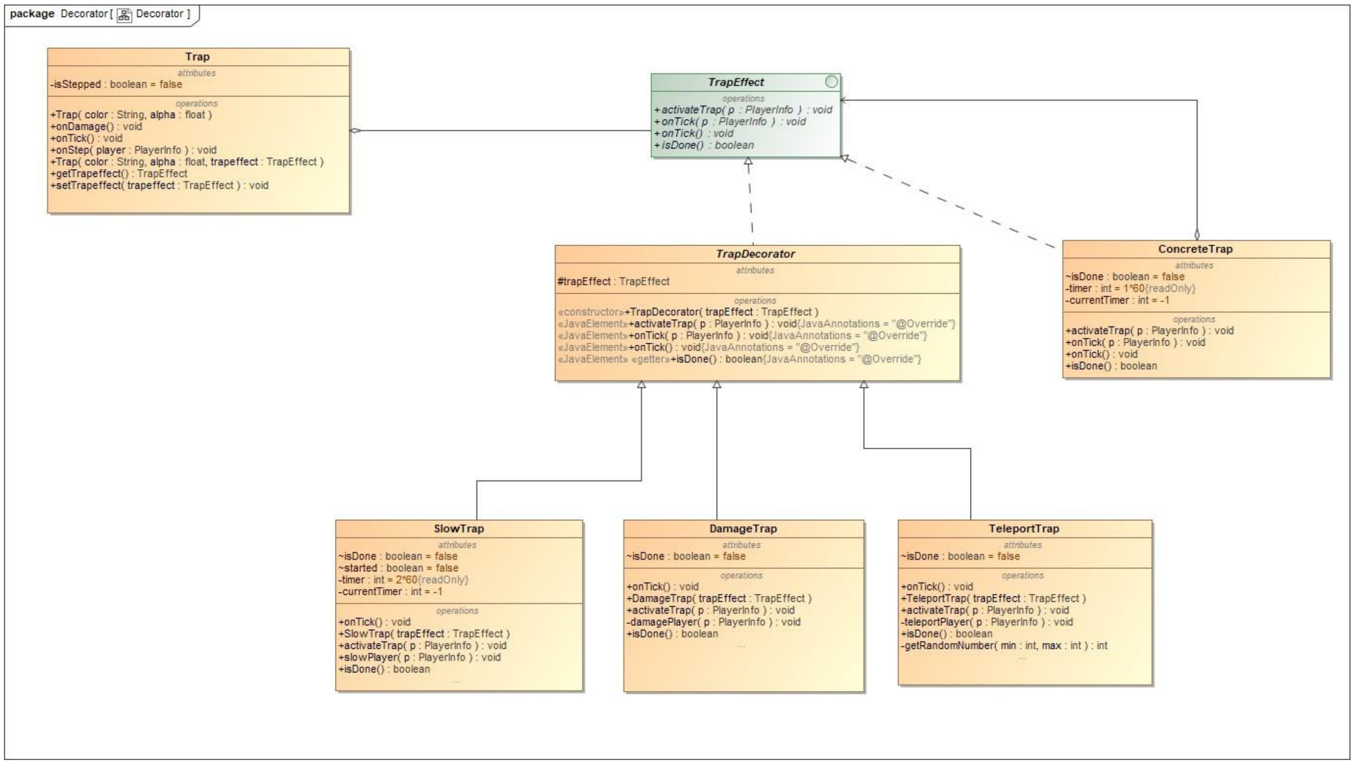
```
  }
}
```

```
package server;

import shared.SimplifiedPlayer;

public class DashSkill implements SkillAlgorithm
{

  private final int cooldown = 10 * 60; //10 seconds
  private final String name = "Dash";
  private int currentCooldown = 0;
  private final int timer = 1 * 30; //0.25 second
  private int currentTimer = 0;
  private SimplifiedPlayer simplified; //For direction
  private float previousSpeed = 0;
  @Override
  public void useSkill(PlayerInfo p)
  {
    if (this.currentCooldown == 0)
    {
      this.currentTimer = timer;
      this.currentCooldown = this.cooldown;
      this.previousSpeed = p.speed;
      p.speed += 10;
      this.simplified = new SimplifiedPlayer();
      this.simplified.isHoldingDown = p.isHoldingDown;
      this.simplified.isHoldingUp = p.isHoldingUp;
      this.simplified.isHoldingLeft = p.isHoldingLeft;
      this.simplified.isHoldingRight = p.isHoldingRight;
    }
}

@Override
public void onTick(PlayerInfo p)
{
    if (this.currentCooldown > 0)
    {
      this.currentCooldown--;
    }

    if (this.currentTimer > 1)
    {
      p.isHoldingDown = this.simplified.isHoldingDown;
      p.isHoldingUp = this.simplified.isHoldingUp;
      p.isHoldingLeft = this.simplified.isHoldingLeft;
      p.isHoldingRight = this.simplified.isHoldingRight;
      this.currentTimer--;
    }
    else if (this.currentTimer == 1)
    {
      p.speed = this.previousSpeed;
      this.currentTimer--;
    }

}

```

```java
    @Override
    public String getName()
    {
        return this.name;
    }

    @Override
    public int getCooldown()
    {
        return this.currentCooldown;
    }
}
```

```java
package server;


public class TeleportSkill implements SkillAlgorithm
{

    private final int cooldown = 10 * 60; //10 seconds
    private final String name = "Teleport";
    private int currentCooldown = 0;

    @Override
    public void useSkill(PlayerInfo p)
    {
        if (this.currentCooldown == 0)
        {
            GameServer gameserver = GameServer.getInstance();
            boolean teleported = false;
            int maxRetry = 60;
            int retry = 0;
            while (!teleported && retry < maxRetry)
            {
                int randomCoordX = getRandomNumber(0, gameserver.gameBoard.
        gridSize);
                int randomCoordY = getRandomNumber(0, gameserver.gameBoard.
        gridSize);
                if (gameserver.gameBoard.objects[randomCoordX][randomCoordY]
        instanceof Ground)
                {
                    p.coordinate.x = randomCoordX * (gameserver.gameBoard.size /
        gameserver.gameBoard.gridSize);
                    p.coordinate.y = randomCoordY * (gameserver.gameBoard.size /
        gameserver.gameBoard.gridSize);
                    teleported = true;
                }
                retry++;
            }
            this.currentCooldown = this.cooldown;
        }
    }

    @Override
    public void onTick(PlayerInfo p)
    {
        if (this.currentCooldown > 0)
        {
            this.currentCooldown--;
        }
    }

    @Override
    public String getName()
    {
        return this.name;
    }

    @Override
    public int getCooldown()
    {
        return this.currentCooldown;
    }

    private int getRandomNumber(int min, int max)
    {
            return (int) ((Math.random() * (max - min)) + min);
    }

}
```

5.1.3. Decorator


package Decorator [Decorator]
Trap
attributes
-isStepped : boolean = false
operations
+Trap(color : String, alpha : float)
+onDamage() : void
+onTick() : void
+onStep(player : PlayerInfo) : void
+Trap(color : String, alpha : float, trapeffect : TrapEffect)
+getTrapeffect() : TrapEffect
+setTrapeffect(trapeffect : TrapEffect) : void
TrapEffect
operations
+activateTrap(p : PlayerInfo) : void
+onTick(p : PlayerInfo) : void
+onTick() : void
+isDone() : boolean
TrapDecorator
attributes
#trapEffect : TrapEffect
operations
«constructor»+TrapDecorator(trapEffect : TrapEffect)
«JavaElement»+activateTrap(p : PlayerInfo) : void{JavaAnnotations = "@Override"}
«JavaElement»+onTick(p : PlayerInfo) : void{JavaAnnotations = "@Override"}
«JavaElement»+onTick() : void{JavaAnnotations = "@Override"}
«JavaElement» «getter»+isDone() : boolean{JavaAnnotations = "@Override"}
ConcreteTrap
attributes
~isDone : boolean = false
-timer : int = 1*60{readOnly}
-currentTimer : int = -1
operations
+activateTrap(p : PlayerInfo) : void
+onTick(p : PlayerInfo) : void
+onTick() : void
+isDone() : boolean
SlowTrap
attributes
~isDone : boolean = false
~started : boolean = false
-timer : int = 2*60{readOnly}
-currentTimer : int = -1
operations
+onTick() : void
+SlowTrap(trapEffect : TrapEffect)
+activateTrap(p : PlayerInfo) : void
+slowPlayer(p : PlayerInfo) : void
+isDone() : boolean
DamageTrap
attributes
~isDone : boolean = false
operations
+onTick() : void
+DamageTrap(trapEffect : TrapEffect)
+activateTrap(p : PlayerInfo) : void
-damagePlayer(p : PlayerInfo) : void
+isDone() : boolean
TeleportTrap
attributes
~isDone : boolean = false
operations
+onTick() : void
+TeleportTrap(trapEffect : TrapEffect)
+activateTrap(p : PlayerInfo) : void
-teleportPlayer(p : PlayerInfo) : void
+isDone() : boolean
-getRandomNumber(min : int, max : int) : int

5.6 pav. Decorator diagrama

Decorator šablono poreikis atsirado norint turėti skirtingus spąstus, su skirtingais efektais, kurie atliekami žaidėjui. Šie efektai gali sluoksniuotis.

```
package server;

public class Trap extends GameObject{


  private TrapEffect trapeffect;
  private boolean isStepped = false;

  public Trap(String color, float alpha, TrapEffect trapeffect)
    {
        super(color, alpha);
        this.trapeffect = trapeffect;
    }

    public void onDamage()
    {
        if(this.isDestroyable)
          isDead = true;
    }
    public void onTick()
    {
      this.trapeffect.onTick();
      if (this.trapeffect.isDone())
      {
        isDead = true;
      }
    }

    public void onStep(PlayerInfo player)
    {
      if (!isStepped)
      {
        this.trapeffect.activateTrap(player);
        isStepped = true;
      }
    }

    public TrapEffect getTrapeffect()
    {
```

```
    return trapeffect;
  }

  public void setTrapeffect(TrapEffect trapeffect)
  {
    this.trapeffect = trapeffect;
  }

}
```

```
package server;

public interface TrapEffect
{
  void activateTrap(PlayerInfo p);
  void onTick();
  boolean isDone();
}
```

```
package server;

public class ConcreteTrap implements TrapEffect
{
  private PlayerInfo p;
  boolean isDone = false;
  private final int timer = 1 * 60; //second
  private int currentTimer =-1;
  @Override
  public void activateTrap(PlayerInfo p)
  {
        this.p = p;
        this.p.speed = 0;
        GameServer gameserver =GameServer.getInstance();
        gameserver.players.get(this.p.id).speed = this.p.speed;t
        his.currentTimer = timer;
    }

    @Override
    public void onTick()
    {

        if (this.currentTimer > 1)
        {
            this.currentTimer--;
        }
        else if (this.currentTimer == 1)
        {
            GameServer gameserver = GameServer.getInstance();
            gameserver.players.get(this.p.id).speed = this.p.baseSpeed;
            this.currentTimer--;
        }
        else if (this.currentTimer == 0)
        {
            this.isDone = true;
        }

    }

    @Override
```

```java
    public boolean isDone()
    {
        return isDone;
    }

}
```

```java
package server;

public abstract class TrapDecorator implements TrapEffect
{
    protected TrapEffect trapEffect;

    public TrapDecorator(TrapEffect trapEffect)
    {
        this.trapEffect = trapEffect;
    }

    @Override
    public void activateTrap(PlayerInfo p)
    {
        this.trapEffect.activateTrap(p);
    }

    @Override
    public void onTick()
    {
        this.trapEffect.onTick();
    }
```

```

  @Override
  public boolean isDone()
  {
    return this.trapEffect.isDone();
  }

}
```

```
package server;

public class SlowTrap extends TrapDecorator
{

  boolean isDone =false;
  boolean started = false;
  private PlayerInfo p;
  private final int timer = 2 * 60; //4 seconds
  private int currentTimer = -1;
  public SlowTrap(TrapEffect trapEffect)
  {
    super(trapEffect);
  }

  public void activateTrap(PlayerInfo p)
  {
    super.activateTrap(p);
    this.p=p;
  }
```

```

public void onTick()
{
  super.onTick();
  slowPlayer(p);
}

public void slowPlayer(PlayerInfo p)
{
  if (super.isDone() && !started)
  {
    this.p.speed = 0;
    GameServer gameserver = GameServer.getInstance();
    gameserver.players.get(this.p.id).speed =  this.p.baseSpeed * 0.5
 f;
    this.currentTimer = timer;
    started = true;
  }

  if (this.currentTimer > 1)
  {
    this.currentTimer--;
  }
  else if (this.currentTimer == 1)
  {
    GameServer gameserver = GameServer.getInstance();
    gameserver.players.get(this.p.id).speed = this.p.baseSpeed;
    this.currentTimer--;
  }
```

```java
        else if (this.currentTimer == 0)
        {
            this.isDone = true;
        }
    }
    public boolean isDone()
    {
        return super.isDone() && isDone;
    }
}
```

```java
package server;

public class DamageTrap extends TrapDecorator
{

    boolean isDone = false;
    public DamageTrap(TrapEffect trapEffect)
    {
        super(trapEffect);
    }

    public void activateTrap(PlayerInfo p)
    {
        super.activateTrap(p);
        damagePlayer(p);
    }

    private void damagePlayer(PlayerInfo p)
```

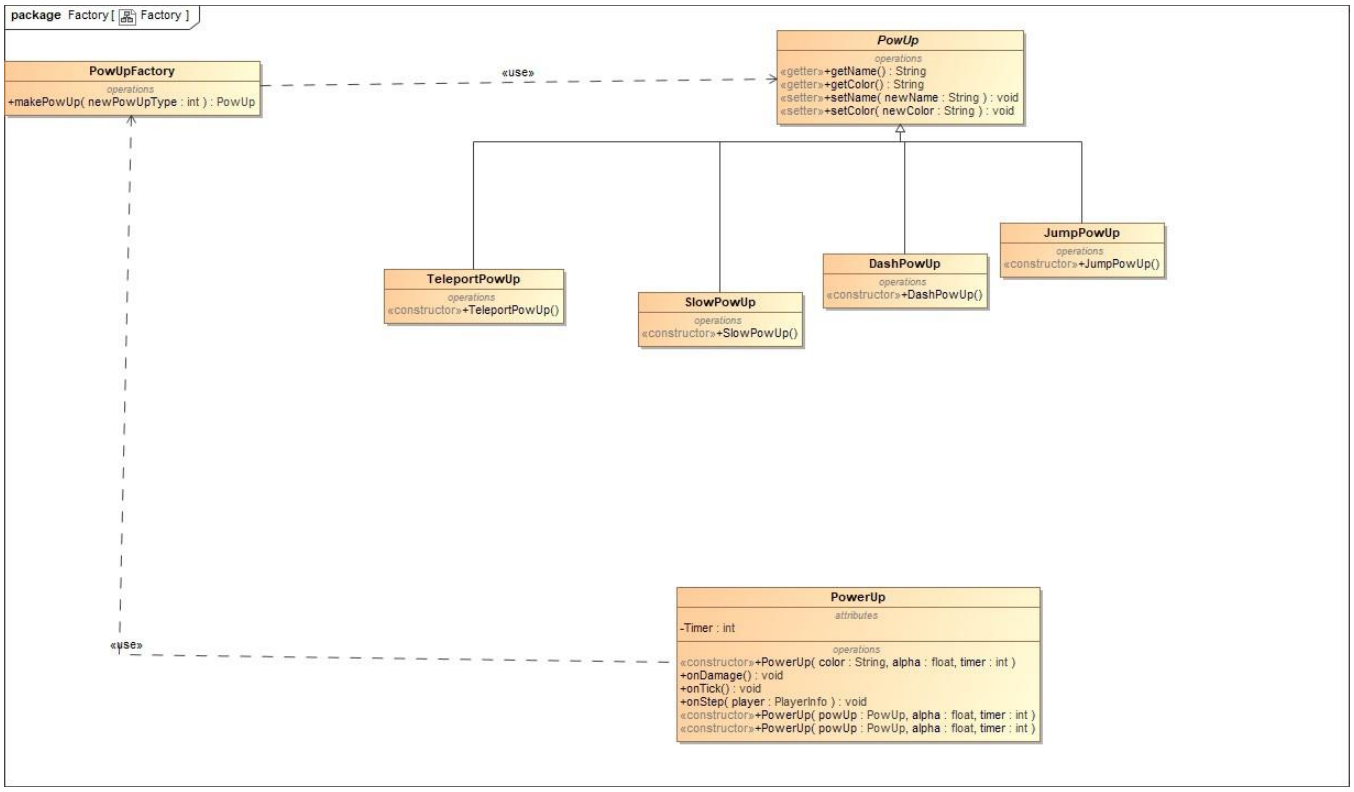
```java
  {
     GameServer gameserver = GameServer.getInstance();
     gameserver.players.get(p.id).health --;
     isDone = true;
  }

  public void onTick()
  {
     super.onTick();
  }

  @Override
   public boolean isDone()
  {
     return super.isDone() && isDone;
  }
}
```

```java
package server;

public class TeleportTrap extends TrapDecorator
{

  boolean isDone = false;
  public TeleportTrap(TrapEffect trapEffect)
  {
     super(trapEffect);
  }

```

```
public void activateTrap(PlayerInfo p)
{
  super.activateTrap(p);
  teleportPlayer(p);

}

private void teleportPlayer(PlayerInfo p)
{
  GameServer gameserver = GameServer.getInstance();
  boolean teleported = false;
  int maxRetry = 60;
  int retry = 0;
  while (!teleported && retry < maxRetry)
  {
    int randomCoordX = getRandomNumber (0, gameserver.gameBoard.
 gridSize);
    int randomCoordY = getRandomNumber (0, gameserver.gameBoard.
 gridSize);
    if (gameserver.gameBoard.objects[randomCoordX][randomCoordY]
 instanceof Ground)
    {
      p.coordinate.x = randomCoordX * (gameserver.gameBoard.size /
 gameserver.gameBoard.gridSize);
      p.coordinate.y = randomCoordY * (gameserver.gameBoard.size /
 gameserver.gameBoard.gridSize);
      teleported = true;
    }
    retry ++;
```

```
        }
        isDone = true;
    }

    public void onTick()
    {
        super.onTick();
    }

    @Override
    public boolean isDone()
    {
        return super.isDone() && isDone;
    }

    private int getRandomNumber(int min, int max)
    {
            return (int) ((Math.random() * (max - min)) + min);
    }
}
```

5.1.4. Abstract Factory


package Factory[Factory]
PowUpFactory
operations
+makePowUp(newPowUpType : int) : PowUp
«use»
PowUp
operations
«getter»+getName() : String
«getter»+getColor() : String
«setter»+setName(newName : String) : void
«setter»+setColor(newColor : String) : void
TeleportPowUp
«constructor»+TeleportPowUp()
SlowPowUp
«constructor»+SlowPowUp()
DashPowUp
«constructor»+DashPowUp()
JumpPowUp
«constructor»+JumpPowUp()
PowerUp
attributes
-Timer : int
«constructor»+PowerUp(color : String, alpha : float, timer : int)
+onDamage() : void
+onTick() : void
+onStep(player : PlayerInfo) : void
«constructor»+PowerUp(powUp : PowUp, alpha : float, timer : int)
«constructor»+PowerUp(powUp : PowUp, alpha : float, timer : int)

5.7 pav. Abstract Factory diagrama

Abstract Factory šablonas padeda lansksčiai sukurti skirtingus žaidimo objektų rinkinius skirtingiems lygiams.

```
package server;

public abstract class AbstractFactory
{
    protected BombObserver bombObserver ;
    public abstract GameObject createTrap();
    public abstract GameObject createWall(boolean destroyable);
    public abstract GameObject createBomb(int ownerid);
    public abstract GameObject createPowerUp();
    public abstract GameObject createGround();
```

```java

    public void SetBombObserver(BombObserver observer)
    {
        this.bombObserver = observer;
    }
}
```

```java
package server;


import java.util.Hashtable;
import java.util.Random;

public class Stage1Factory extends AbstractFactory{

    private Hashtable<String, String> colors;

    public Stage1Factory(){
        this.colors = new Hashtable<>();
        this.colors.put("Wall", "#2e2203");
        this.colors.put("DesWall", "#8c8674");
        this.colors.put("Ground", "#348C31");
        this.colors.put("Bomb", "#6A2E35");
        this.colors.put("PowerUp", "#BAD094");
        this.colors.put("Trap","#373D20");
    }

    @Override
    public GameObject createTrap()
    {
        return new Trap(this.colors.get("Trap"), 1, new ConcreteTrap())
;
    }

    @Override
    public GameObject createWall(boolean destroyable)
    {
        String wallType = "Wall";

        if(destroyable)
            wallType = "DesWall";

        return new Wall(this.colors.get(wallType), 1, destroyable);
    }

    @Override
    public GameObject createBomb(int ownerid) {
        return new Bomb(colors.get("Bomb"), 1, 90, ownerid, this.
bombObserver);
    }

    @Override
    public GameObject createPowerUp() {

        PowUpFactory powFactory = new PowUpFactory();
        PowUp powUp = null;

        Random rand = new Random();
```

```
            int randomNum = rand.nextInt((4 - 1) + 1) + 1;

            powUp = powFactory.makePowUp(randomNum);


            return new PowerUp(powUp, 1, 5);
    }

    @Override
    public GameObject createGround()
    {
        return new Ground(colors.get("Ground"), 1);
    }

}
```

```
package server;


import java.util.Hashtable;
import java.util.Random;

public class Stage2Factory extends AbstractFactory{

    private Hashtable<String, String> colors;

    public Stage2Factory(){
        this.colors = new Hashtable<>();
        this.colors.put("Wall", "#ECE5F0");
        this.colors.put("DesWall", "#CEB5A7");
        this.colors.put("Ground", "#FFA3AF");
        this.colors.put("Bomb", "#210124");
        this.colors.put("PowerUp", "#87FF65");
        this.colors.put("Trap", "#8EA604");
    }

    @Override
    public GameObject createTrap()
    {
        return new Trap(this.colors.get("Trap"), 1, new ConcreteTrap())
;
    }

    @Override
    public GameObject createWall(boolean destroyable)
    {
        String wallType = "Wall";

        if(destroyable)
            wallType = "DesWall";

        return new Wall(this.colors.get(wallType), 1, destroyable);
    }

    @Override
    public GameObject createBomb(int ownerid) {
        return new Bomb(colors.get("Bomb"), 1, 90, ownerid, this.
bombObserver);
```

```
    }

    @Override
    public GameObject createPowerUp() {

        PowUpFactory powFactory = new PowUpFactory();
        PowUp powUp = null;

        Random rand = new Random();
        int randomNum = rand.nextInt((4 - 1) + 1) + 1;

        powUp = powFactory.makePowUp(randomNum);


        return new PowerUp(powUp, 1, 5);
    }

    @Override
    public GameObject createGround()
    {
      return new Ground(colors.get("Ground"), 1);
    }

}
```

```
package server;

public abstract class TrapDecorator implements TrapEffect
{
```

```java
    protected TrapEffect trapEffect;

    public TrapDecorator(TrapEffect trapEffect)
    {
        this.trapEffect = trapEffect;
    }

    @Override
    public void activateTrap(PlayerInfo p)
    {
        this.trapEffect.activateTrap(p);
    }

    @Override
    public void onTick()
    {
        this.trapEffect.onTick();
    }

    @Override
    public boolean isDone()
    {
        return this.trapEffect.isDone();
    }

}
```

```java
package server;
import java.util.Random;
```

```

public class PowerUp extends GameObject
{
      private int Timer;
   private PowUp powUp;

      public PowerUp(PowUp powUp, float alpha, int timer){


            super(powUp.getColor(), alpha);
      this.powUp = powUp;
            this.Timer = timer;
            this.isWalkable = true;



      }

      public void onDamage(){
            if(this.isDestroyable)
                  System.out.println("Wall has been destroyed");
      }
      public void onTick(){
//             this.Timer--;
//             if(this.Timer <= 0)
//                isDead = true;
      }


```

```
  public void onStep(PlayerInfo player){
        if(this.isWalkable)
        {
           if (!isDead)
           {
        GameServer gameServer = GameServer.getInstance();


        if(powUp.getName().equals("Jump"))
           gameServer.players.get(player.id).setSkillAlgorithm(new
    JumpSkill());

        else if(powUp.getName().equals("Dash"))
           gameServer.players.get(player.id).setSkillAlgorithm(new
    DashSkill());

        else if(powUp.getName().equals("Slow"))
           gameServer.players.get(player.id).setSkillAlgorithm(new
    SlowAllPlayersSkill());

        else if(powUp.getName().equals("Teleport"))
           gameServer.players.get(player.id).setSkillAlgorithm(new
    TeleportSkill());
```

```java




                    isDead = true;
                }
            }
                    System.out.println("You gain powerup!");
    }
}
```

```java
package server;

public class Ground extends GameObject{

    public Ground(String color, float alpha){
        super(color, alpha );
        this.isWalkable = true;
    }

    @Override
    public void onDamage() {

    }

    @Override
    public void onTick() {

    }

    @Override
    public void onStep(PlayerInfo player) {

    }
}
```

```java
package server;

import java.util.List;
import java.util.Stack;

public class Bomb extends GameObject implements BombObservable{

    private int Timer;
    private int OwnerId;
    private int ExplosionRadius = 3;
    private List<BombObserver> observers = new Stack<BombObserver>();

    public Bomb(String color, float alpha, int timer, int ownerid,
  BombObserver observer){
        super(color, alpha);
        this.Timer = timer;
        this.OwnerId = ownerid;
        this.add(observer);
        this.isWalkable = true;
    }

    public void onDamage(){
        //cannot be destroyed?
    }
    public void onTick(){
        this.Timer--;
        if(this.Timer <= 0)
            notifyObservers();
    }

    public void onStep(PlayerInfo player){
        //cannot be stepped on?
    }

    public void add(BombObserver observer)
    {
        this.observers.add(observer);
    }

    public void remove(BombObserver observer)
    {
        this.observers.remove(observer);
    }

    public void notifyObservers()
    {
        for (BombObserver observer :
             observers) {
            observer.explode(this);

        }
```

```java
    }

    public int explosionRadius() {
        if (this.Timer <= 0) return this.ExplosionRadius;
        else return 0;
    }
}
```

```java
package server;

public class Wall extends GameObject{


    public Wall(String color, float alpha, boolean destroyable){
        super(color, alpha);
        this.isDestroyable = destroyable;
    }

    public void onDamage()
    {
        if(this.isDestroyable)
            isDead = true;
    }

    public void onTick()
    {

    }

    public void onStep(PlayerInfo player)
    {
        if(!this.isWalkable)
            System.out.println("Can't walk on this!");
    }
}
```

```
package server;

public class Trap extends GameObject{


  private TrapEffect trapeffect;
  private boolean isStepped = false;

  public Trap(String color, float alpha, TrapEffect trapeffect)
    {
        super(color, alpha);
        this.trapeffect = trapeffect;
    }

    public void onDamage()
    {
        if(this.isDestroyable)
          isDead = true;
    }
    public void onTick()
    {
      this.trapeffect.onTick();
```

```
        if (this.trapeffect.isDone())
        {
          isDead = true;
        }
      }

      public void onStep(PlayerInfo player)
      {
        if (!isStepped)
        {
          this.trapeffect.activateTrap(player);
          isStepped = true;
        }
      }

      public TrapEffect getTrapeffect()
      {
      return trapeffect;
    }

    public void setTrapeffect(TrapEffect trapeffect)
    {
      this.trapeffect = trapeffect;
    }

}
```

```
package server;

```

```
public abstract class GameObject {
    public boolean isWalkable;
    public boolean isDestroyable;
    public boolean isDead;

    GameObjectDelegate gameobjectdelegate;

    public String color;
    public float alpha;

    public GameObject(String color, float alpha) {
        super();
        this.alpha = alpha;
        this.color = color;
    }

    public void sayHello() {
      System.out.println("GameObject");
    }

    public void setDestroyable(boolean option){
        this.isDestroyable = option;
    }

    public void setWalkable(boolean option){
        this.isWalkable = option;
    }

    public abstract void onDamage();
    public abstract void onTick();
    public abstract void onStep(PlayerInfo player);

}
```

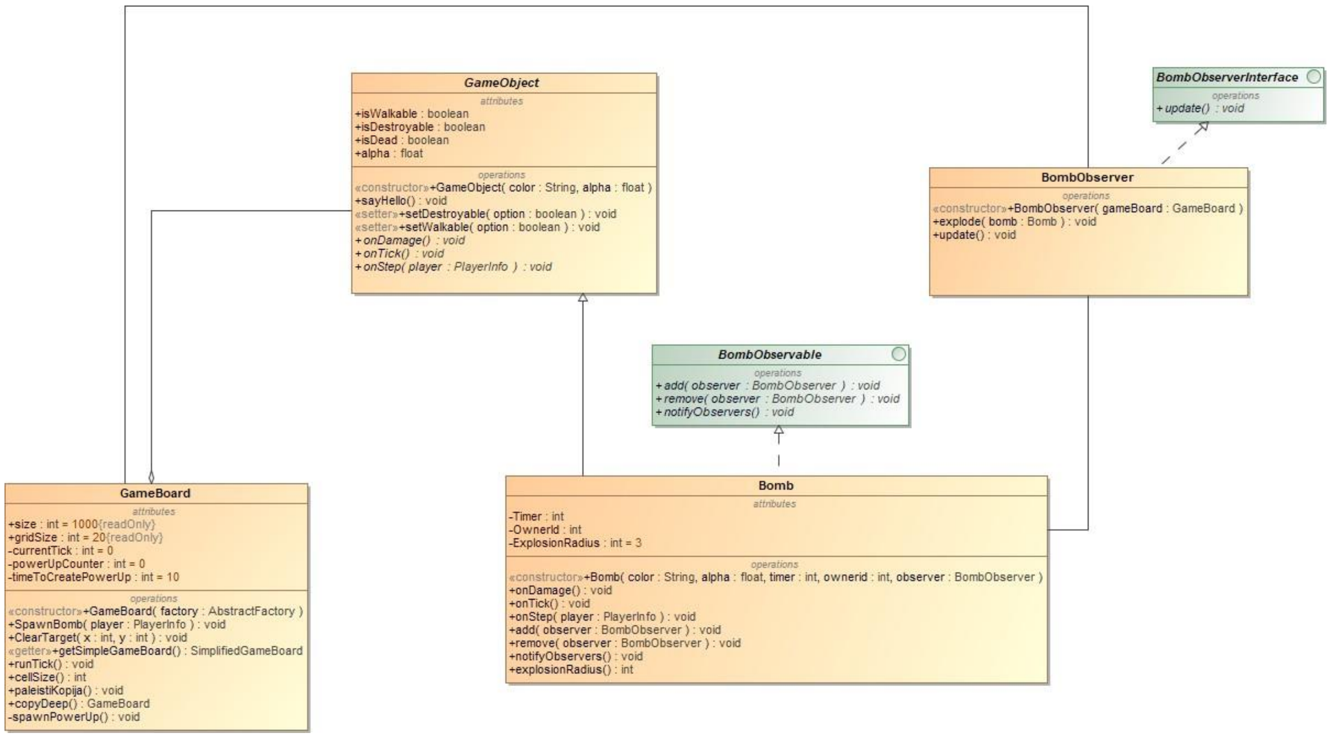
```
package server;

import java.util.ArrayList;
import java.util.Random;

import shared.*;

public class GameBoard implements Cloneable{

    public final int size = 1000;
    public final int gridSize = 20;
    public GameObject[][] objects;
    private BombObserver bombObserver;
    private AbstractFactory factory;

    private int currentTick = 0;
    private int powerUpCounter = 0;
    private int timeToCreatePowerUp = 10;

    private IStageBuilder stage1builder;
    private IStageBuilder stage2builder;
    private IStageBuilder stage3builder;

    public GameBoard(AbstractFactory factory)
```

```
  {
      this.factory = factory;
      this.objects = new GameObject[gridSize][gridSize];
      this.stage1builder = new Stage1Builder(gridSize);
      this.stage2builder = new Stage2Builder(gridSize);
      this.stage3builder = new Stage3Builder(gridSize);

      StageDirector stageDirector = new StageDirector(stage3builder);

      stageDirector.makeStage();

      Stage stage1 = stageDirector.getStage();

      for(Coordinates kor: stage1.getGrounds())
          this.objects[kor.getX()][kor.getY()]=this.factory.
createGround();

      int kiekis = 76;
      boolean sunaikinama = false;

      for(Coordinates kor: stage1.getWalls()){

          this.objects[kor.getX()][kor.getY()]=this.factory.
createWall(sunaikinama);
          kiekis -=1;
          if(kiekis == 0) sunaikinama = true;
      }


```

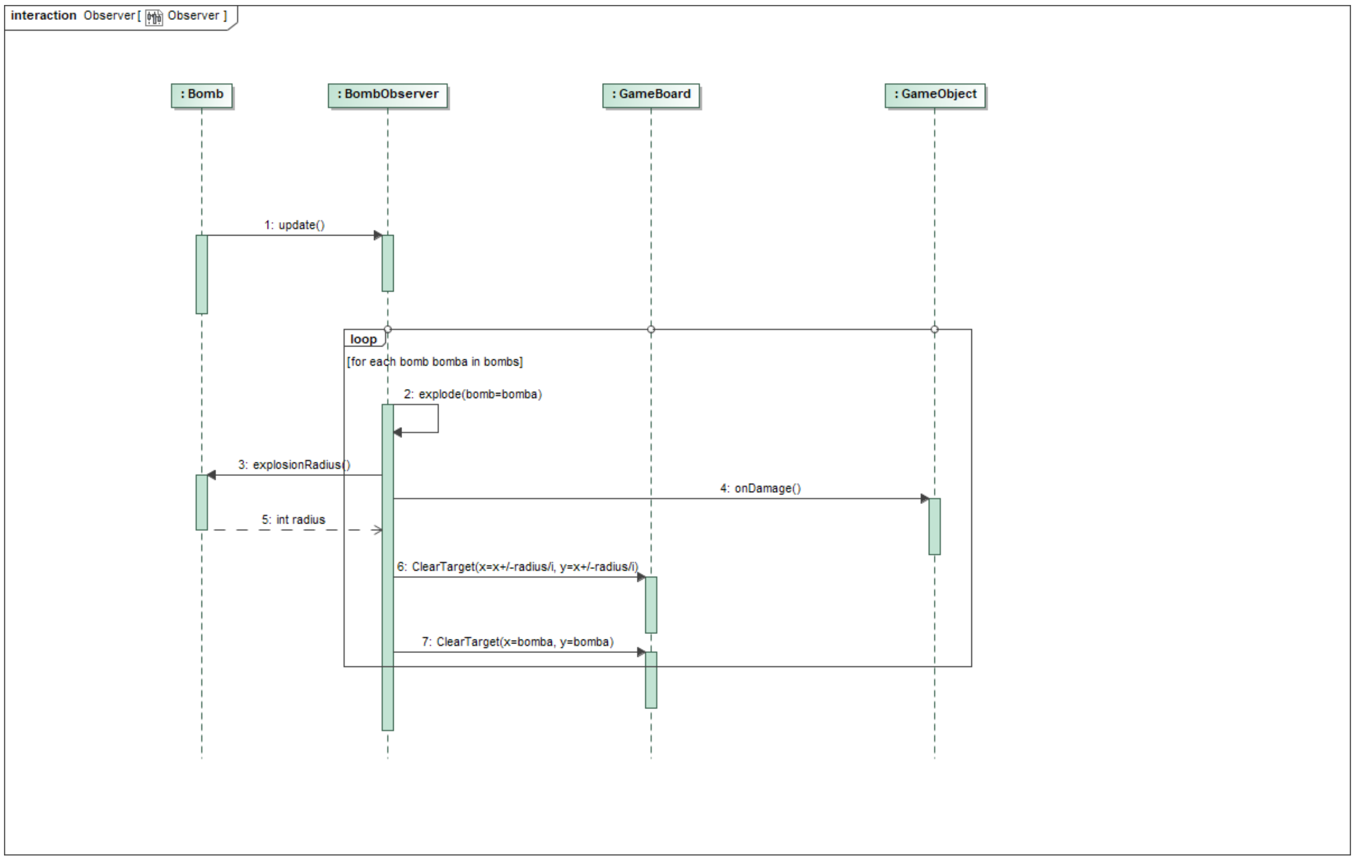
```

        bombObserver = new BombObserver(this);


        this.factory.SetBombObserver(bombObserver);





        //test

        //TODO wrong but ok for now
        this.objects[17][3] = this.factory.createTrap();

        this.objects[15][3] = this.factory.createTrap();
        Trap modified = (Trap) this.objects[15][3];
        modified.setTrapeffect(new DamageTrap (new ConcreteTrap()));
        this.objects[15][3] = modified;

        this.objects[13][3] = this.factory.createTrap();
        Trap modified1 = (Trap) this.objects[13][3];
        modified1.setTrapeffect(new TeleportTrap (new ConcreteTrap()));
        this.objects[13][3] = modified1;

        this.objects[11][3] = this.factory.createTrap();
        Trap modified2 = (Trap) this.objects[11][3];
        modified2.setTrapeffect(new SlowTrap (new ConcreteTrap()));
        this.objects[11][3] = modified2;

        this.objects[9][3] = this.factory.createTrap();
        Trap modified3 = (Trap) this.objects[9][3];
        modified3.setTrapeffect(new DamageTrap(new TeleportTrap (new
    SlowTrap (new ConcreteTrap())))) ;
        this.objects[9][3] = modified3;

    }

    public void paleistiKopija(){
        GameBoard copy = copyDeep();


    }



    public GameBoard copyDeep(){

        try {
            return (GameBoard) this.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
            return null;
        }
    }

    public void SpawnBomb(PlayerInfo player)
    {
        int x = Math.round(player.coordinate.x/(size/gridSize));
        int y = Math.round(player.coordinate.y/(size/gridSize));
        this.objects[x][y] = this.factory.createBomb(player.id);

    }

    public void ClearTarget(int x, int y){
        this.objects[x][y] = this.factory.createGround();
        System.out.println("Removing bomb from location " + x + " " + y
);
    }

    public SimplifiedGameBoard getSimpleGameBoard()
    {
    SimplifiedGameBoard simpleGameboard = new SimplifiedGameBoard(this.
size, this.gridSize);

        for (int i = 0; i < this.gridSize; i ++)
        {
            for (int j = 0; j < this.gridSize; j ++)
            {
                simpleGameboard.objects[i][j] = new SimplifiedGameObject
(this.objects[i][j].color, ObjectType.GROUND);
            }
        }
        return simpleGameboard;
    }

    private void spawnPowerUp(){
        Random rand = new Random();

        if(powerUpCounter > 4)
            return;

        while(true){
            int x = rand.nextInt((19 - 0) + 1) + 0;
            int y = rand.nextInt((19 - 0) + 1) + 0;


            if(this.objects[x][y] instanceof Ground){
                this.objects[x][y] = this.factory.createPowerUp();
                powerUpCounter+=1;
                break;
            }

        }








    }


    public void runTick()
```

```
  {
      currentTick  +=1;

      if(currentTick % (60 * timeToCreatePowerUp)== 0) // kas 10s
sukurti nauja powerUp
            spawnPowerUp() ;

      for(int x = 0; x < this.gridSize; x++)
      {
            for (int y = 0; y < this.gridSize; y++)
            {
               if(this.objects[x][y].isDead) {



                        if(this.objects[x][y] instanceof PowerUp)
                        {
                              System.out.println("Sunaikinau powerUpa");
                              powerUpCounter -=1;
                        }

                        this.ClearTarget(x, y);



                  }

               objects[x][y].onTick();
            }
            }
        }

        public int cellSize()
        {
            return size / gridSize;
        }
}
```

5.1.5. Observer


GameObject
attributes
+isWalkable : boolean
+isDestroyable : boolean
+isDead : boolean
+alpha : float
operations
«constructor»+GameObject(color : String, alpha : float)
+sayHello() : void
«setter»+setDestroyable(option : boolean) : void
«setter»+setWalkable(option : boolean) : void
+onDamage() : void
+onTick() : void
+onStep(player : PlayerInfo) : void
BombObserverInterface
operations
+update() : void
BombObserver
operations
«constructor»+BombObserver(gameBoard : GameBoard)
+explode(bomb : Bomb) : void
+update() : void
BombObservable
operations
+add(observer : BombObserver) : void
+remove(observer : BombObserver) : void
+notifyObservers() : void
GameBoard
attributes
+size : int = 1000{readOnly}
+gridSize : int = 20{readOnly}
-currentTick : int = 0
-powerUpCounter : int = 0
-timeToCreatePowerUp : int = 10
operations
«constructor»+GameBoard(factory : AbstractFactory)
+SpawnBomb(player : PlayerInfo) : void
+ClearTarget(x : int, y : int) : void
«getter»+getSimpleGameBoard() : SimplifiedGameBoard
+runTick() : void
+cellSize() : int
+paleistiKopija() : void
+copyDeep() : GameBoard
-spawnPowerUp() : void
Bomb
attributes
-Timer : int
-OwnerId : int
-ExplosionRadius : int = 3
operations
«constructor»+Bomb(color : String, alpha : float, timer : int, ownerid : int, observer : BombObserver)
+onDamage() : void
+onTick() : void
+onStep(player : PlayerInfo) : void
+add(observer : BombObserver) : void
+remove(observer : BombObserver) : void
+notifyObservers() : void
+explosionRadius() : int

5.8 pav. Observer diagrama


interaction Observer [Observer]
: Bomb
: BombObserver
: GameBoard
: GameObject
1: update()
loop
[for each bomb bomba in bombs]
2: explode(bomb=bomba)
3: explosionRadius()
4: onDamage()
5: int radius
6: ClearTarget(x=x+/-radius/i, y=x+/-radius/i)
7: ClearTarget(x=bomba, y=bomba)

5.9 pav. Observer usecase

Observer optimaliai apdoroja bombos informacija tik tada, kai bomba yra pasiruošusi sprogti.

```java
package server;

import java.util.List;
import java.util.Stack;

public class Bomb extends GameObject implements BombObservable{

    private int Timer ;
    private int OwnerId;
    private int ExplosionRadius = 3;
    private List<BombObserver> observers = new Stack<BombObserver>();

    public Bomb(String color, float alpha, int timer, int ownerid,
  BombObserver observer){
        super(color, alpha);
        this.Timer = timer;
        this.OwnerId = ownerid;
        this.add(observer);
        this.isWalkable = true;
    }

    public void onDamage(){
        //cannot be destroyed?
    }
    public void onTick(){
        this.Timer--;
        if(this.Timer <= 0)
            notifyObservers();
    }

    public void onStep(PlayerInfo player){
        //cannot be stepped on?
    }

    public void add(BombObserver observer)
    {
        this.observers.add(observer);
    }

    public void remove(BombObserver observer)
    {
            this.observers.remove(observer);
    }

    public void notifyObservers()
    {
        for (BombObserver observer :
             observers){
            observer.explode(this);

        }
    }

    public int explosionRadius() {
        if (this.Timer <= 0) return this.ExplosionRadius;
        else return 0;
    }
}
```

```
package server;

public interface BombObservable {
    public void add(BombObserver observer);
    public void remove (BombObserver observer);
    public void notifyObservers ();

}
```

```
package server;

import java.util.List;
```

```java
import java.util.Stack;

public class BombObserver implements BombObserverInterface {
    private GameBoard gameBoard;
    private List<Bomb> bombs = new Stack<Bomb>();


    public BombObserver(GameBoard gameBoard){
        this.gameBoard = gameBoard;
    }

    public void update() {
        for(int x = 0; x < this.gameBoard.gridSize; x++) {
            for (int y = 0; y < this.gameBoard.gridSize; y++) {
                if (this.gameBoard.objects[x][y].getClass().equals(Bomb
.class))
                {
                    bombs.add((Bomb) this.gameBoard.objects[x][y]);
                }
            }
        }

        for (Bomb bomb: bombs) {
            if(bomb.explosionRadius()>0){
                explode(bomb);
            }
        }
    }

```

```
  public void explode(Bomb bomb){
      for(int x = 0; x < this.gameBoard.gridSize; x++)
      {
          for (int y = 0; y < this.gameBoard.gridSize; y++)
          {
              if (this.gameBoard.objects[x][y] == bomb)
              {
                  for(int i = 0; i < bomb.explosionRadius(); i++) {
                      //TODO: Make a sophisticated explosion radius
calculation
                      this.gameBoard.ClearTarget(x, y);
                      if(x+ i < this.gameBoard.gridSize) {
                          this.gameBoard.objects[x+ i][y].onDamage()
;
                          if (this.gameBoard.objects[x+ i][y].isDead
) this.gameBoard.ClearTarget(x + 1, y);
                      }
                      if(y+ i < this.gameBoard.gridSize) {
                          this.gameBoard.objects[x][y+ i].onDamage()
;
                          if (this.gameBoard.objects[x][y+ i].isDead
) this.gameBoard.ClearTarget(x, y + 1);
                      }
                      if(x - i > 0) {
                          this.gameBoard.objects[x - i][y].onDamage()
;
                          if (this.gameBoard.objects[x - i][y].isDead
) this.gameBoard.ClearTarget(x - 1, y);
                      }
```

```
                        if(y - i > 0) {
                            this.gameBoard.objects[x][y - i].onDamage()
    ;
                            if (this.gameBoard.objects[x][y - i].isDead
    ) this.gameBoard.ClearTarget(x, y - 1);
                        }
                    }
                }
            }
        }
    }
}
```

```
package server;

public interface BombObserverInterface {
    public void update();
}
```

```
package server;

public abstract class GameObject {
    public boolean isWalkable;
    public boolean isDestroyable;
    public boolean isDead;

    GameObjectDelegate gameobjectdelegate;

    public String color;
    public float alpha;
```

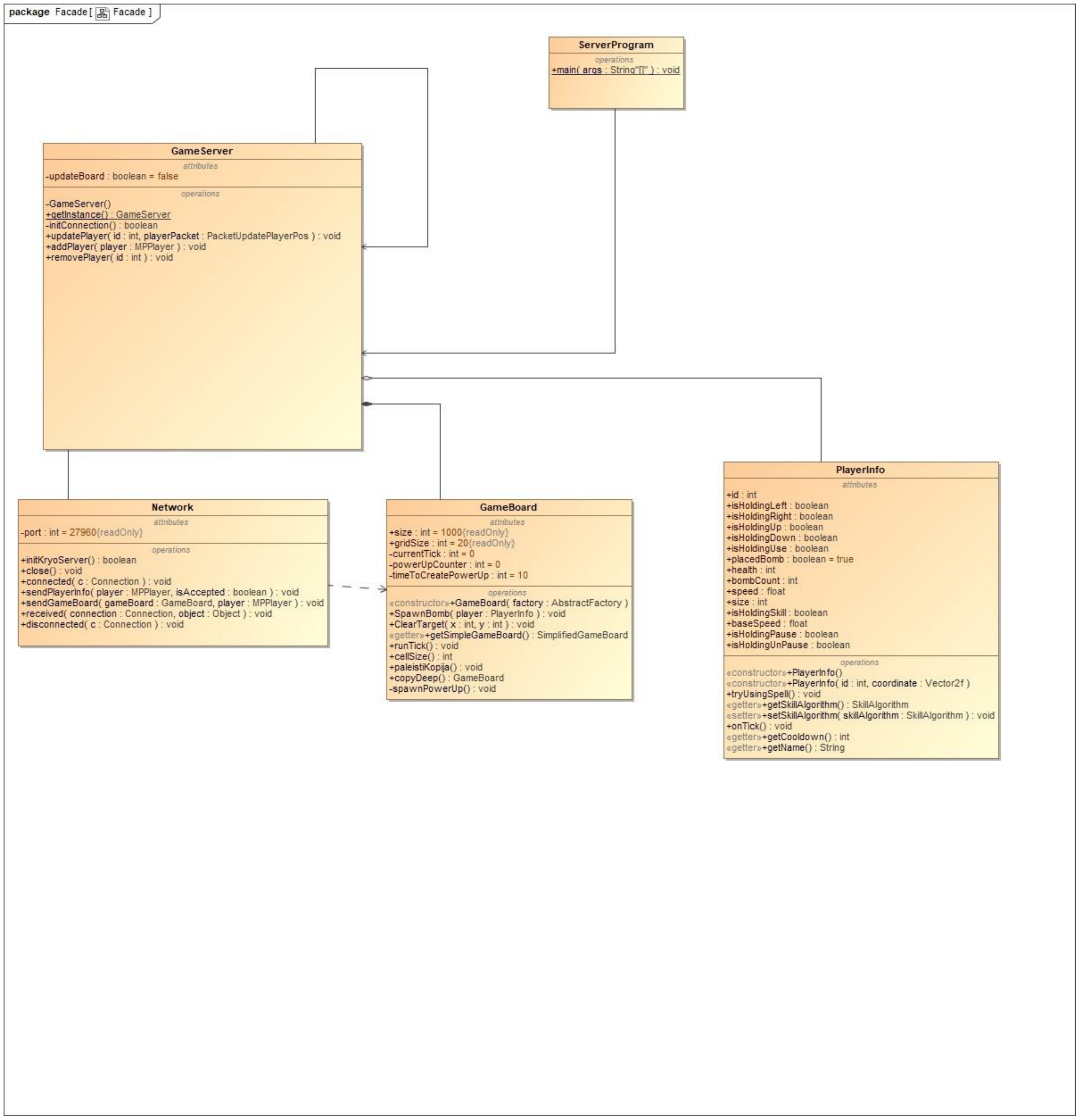
```java

    public GameObject(String color, float alpha) {
        super();
        this.alpha = alpha;
        this.color = color;
    }

    public void sayHello() {
      System.out.println("GameObject");
    }

    public void setDestroyable(boolean option){
        this.isDestroyable = option;
    }

    public void setWalkable(boolean option){
        this.isWalkable = option;
    }

    public abstract void onDamage();
    public abstract void onTick();
    public abstract void onStep(PlayerInfo player);

}
```

```java
package server;

import java.util.ArrayList;
import java.util.Random;

import shared.*;

public class GameBoard implements Cloneable{

    public final int size = 1000;
    public final int gridSize = 20;
    public GameObject[][] objects;
    private BombObserver bombObserver;
    private AbstractFactory factory;

    private int currentTick = 0;
    private int powerUpCounter = 0;
    private int timeToCreatePowerUp = 10;

    private IStageBuilder stage1builder;
    private IStageBuilder stage2builder;
    private IStageBuilder stage3builder;

    public GameBoard(AbstractFactory factory)
    {
        this.factory = factory;
        this.objects = new GameObject[gridSize][gridSize];
        this.stage1builder = new Stage1Builder(gridSize);
        this.stage2builder = new Stage2Builder(gridSize);
        this.stage3builder = new Stage3Builder(gridSize);

        StageDirector stageDirector = new StageDirector(stage3builder);

        stageDirector.makeStage();

        Stage stage1 = stageDirector.getStage();

        for(Coordinates kor: stage1.getGrounds())
            this.objects[kor.getX()][kor.getY()]=this.factory.
createGround();

        int kiekis = 76;
        boolean sunaikinama = false;

        for(Coordinates kor: stage1.getWalls()){

            this.objects[kor.getX()][kor.getY()]=this.factory.
createWall(sunaikinama);
            kiekis -=1;
            if(kiekis == 0) sunaikinama = true;
        }



        bombObserver = new BombObserver(this);


        this.factory.SetBombObserver(bombObserver);





        //test

        //TODO wrong but ok for now
        this.objects[17][3] = this.factory.createTrap();

        this.objects[15][3] = this.factory.createTrap();
        Trap modified = (Trap) this.objects[15][3];
        modified.setTrapeffect(new DamageTrap (new ConcreteTrap()));
        this.objects[15][3] = modified;

        this.objects[13][3] = this.factory.createTrap();
        Trap modified1 = (Trap) this.objects[13][3];
        modified1.setTrapeffect(new TeleportTrap (new ConcreteTrap()));
        this.objects[13][3] = modified1;

        this.objects[11][3] = this.factory.createTrap();
        Trap modified2 = (Trap) this.objects[11][3];
        modified2.setTrapeffect(new SlowTrap (new ConcreteTrap()));
        this.objects[11][3] = modified2;

        this.objects[9][3] = this.factory.createTrap();
        Trap modified3 = (Trap) this.objects[9][3];
        modified3.setTrapeffect( new DamageTrap(new TeleportTrap (new
  SlowTrap (new ConcreteTrap()))));
        this.objects[9][3] = modified3;

    }

    public void paleistiKopija(){
        GameBoard copy = copyDeep();


    }



    public GameBoard copyDeep(){

        try {
            return (GameBoard) this.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
            return null;
        }
    }

    public void SpawnBomb(PlayerInfo player)
    {
        int x = Math.round(player.coordinate.x/(size/gridSize));
        int y = Math.round(player.coordinate.y/(size/gridSize));
        this.objects[x][y] = this.factory.createBomb(player.id);

    }

    public void ClearTarget(int x, int y){
        this.objects[x][y] = this.factory.createGround();
        System.out.println("Removing bomb from location " + x + " " + y
  );
    }

    public SimplifiedGameBoard getSimpleGameBoard()
    {
    SimplifiedGameBoard simpleGameboard = new SimplifiedGameBoard(this.
  size, this.gridSize);

        for (int i = 0; i < this.gridSize; i++)
        {
            for (int j = 0; j < this.gridSize; j++)
            {
              simpleGameboard.objects[i][j] = new SimplifiedGameObject
  (this.objects[i][j].color, ObjectType.GROUND);
            }
        }
        return simpleGameboard;
    }

    private void spawnPowerUp(){
        Random rand = new Random();

        if(powerUpCounter > 4)
            return;

        while(true){
            int x = rand.nextInt((19 - 0) + 1) + 0;
            int y = rand.nextInt((19 - 0) + 1) + 0;

```

```

                if(this.objects[x][y] instanceof Ground){
                    this.objects[x][y] = this.factory.createPowerUp();
                    powerUpCounter+=1;
                    break;
                }

            }








        }


        public void runTick()
        {
            currentTick +=1;

            if(currentTick % (60 * timeToCreatePowerUp)== 0) // kas 10s
    sukurti nauja powerUp
                spawnPowerUp() ;

            for(int x = 0; x < this.gridSize; x++)
            {
```

```
            for (int y = 0; y < this.gridSize; y++)
            {
              if(this.objects[x][y].isDead) {



                    if(this.objects[x][y] instanceof PowerUp)
                    {
                        System.out.println("Sunaikinau powerUpa");
                        powerUpCounter-=1;
                    }

                    this.ClearTarget(x, y);



                }

              objects[x][y].onTick();
            }
        }
    }

    public int cellSize()
    {
      return size / gridSize;
    }
}
```

5.1.6. Facade


package Facade [Facade]
ServerProgram
operations
+main(args : String"[]") : void
GameServer
attributes
-updateBoard : boolean = false
operations
-GameServer()
+getInstance() : GameServer
-initConnection() : boolean
+updatePlayer(id : int, playerPacket : PacketUpdatePlayerPos) : void
+addPlayer(player : MPPlayer) : void
+removePlayer(id : int) : void
Network
attributes
-port : int = 27960{readOnly}
operations
+initKryoServer() : boolean
+close() : void
+connected(c : Connection) : void
+sendPlayerInfo(player : MPPlayer, isAccepted : boolean) : void
+sendGameBoard(gameBoard : GameBoard, player : MPPlayer) : void
+received(connection : Connection, object : Object) : void
+disconnected(c : Connection) : void
GameBoard
attributes
+size : int = 1000{readOnly}
+gridSize : int = 20{readOnly}
-currentTick : int = 0
-powerUpCounter : int = 0
-timeToCreatePowerUp : int = 10
operations
«constructor»+GameBoard(factory : AbstractFactory)
+SpawnBomb(player : PlayerInfo) : void
+ClearTarget(x : int, y : int) : void
«getter»+getSimpleGameBoard() : SimplifiedGameBoard
+runTick() : void
+cellSize() : int
+paleistiKopija() : void
+copyDeep() : GameBoard
-spawnPowerUp() : void
PlayerInfo
attributes
+id : int
+isHoldingLeft : boolean
+isHoldingRight : boolean
+isHoldingUp : boolean
+isHoldingDown : boolean
+isHoldingUse : boolean
+placedBomb : boolean = true
+health : int
+bombCount : int
+speed : float
+size : int
+isHoldingSkill : boolean
+baseSpeed : float
+isHoldingPause : boolean
+isHoldingUnPause : boolean
operations
«constructor»+PlayerInfo()
«constructor»+PlayerInfo(id : int, coordinate : Vector2f)
+tryUsingSpell() : void
«getter»+getSkillAlgorithm() : SkillAlgorithm
«setter»+setSkillAlgorithm(skillAlgorithm : SkillAlgorithm) : void
+onTick() : void
«getter»+getCooldown() : int
«getter»+getName() : String

5.10 pav. Facade diagrama

Naudojant Facade mūsų serverio žaidimo logikos funkcionalumas yra prieinamas prie GameServer klasės kuri yra atitraukta nuo viso kito programos funkcionalumo, taigi jeigu reikėtų implemetuoti papildomas sub sistemas, kaip monitoring arba serverio GUI, tai būtų lengviau.

```java
package server;

public class ServerProgram
{
  public static void main(String[] args)
  {
    GameServer.getInstance();
  }
}
```

```java
package server;

import java.util.HashMap;
import java.util.Map;

import shared.Vector2f;
import shared.PacketUpdatePlayerPos;

class GameServer
{
  private static GameServer gameServer = null;

  protected volatile GameBoard gameBoard;
  protected volatile Map<Integer, MPPlayer> players;
  protected volatile Network network;
  private GameCycleThread thread ;
  private Stage1Factory stage1factory;
  private boolean updateBoard = false;

private GameServer()
{
  //Init Connection
  if (!initConnection())
  {
    System.err.println("ERROR Connecting to host");
    return;
  }

  this.players = new HashMap<Integer, MPPlayer>();
  this.stage1factory = new Stage1Factory();
  this.gameBoard = new GameBoard(stage1factory);

  this.thread = new GameCycleThread();
  this.thread.start();
}

public static GameServer getInstance()
{
  if (gameServer == null)
  {
    gameServer = new GameServer();
  }
  return gameServer;
}

private boolean initConnection()
{
  this.network = new Network();
```

```

    if (!this.network.initKryoServer())
    {
        return false;
    }

    return true;
}

public void updatePlayer(int id, PacketUpdatePlayerPos playerPacket)
{
    MPPlayer player = players.get(id);
    if (player != null)
    {
        player.isHoldingUp = playerPacket.isHoldingUp != null ?
  playerPacket.isHoldingUp : player.isHoldingUp;
        player.isHoldingDown = playerPacket.isHoldingDown != null ?
  playerPacket.isHoldingDown : player.isHoldingDown;
        player.isHoldingLeft = playerPacket.isHoldingLeft != null ?
  playerPacket.isHoldingLeft : player.isHoldingLeft;
        player.isHoldingRight = playerPacket.isHoldingRight != null ?
  playerPacket.isHoldingRight : player.isHoldingRight;
        player.isHoldingUse = playerPacket.isHoldingUse != null ?
  playerPacket.isHoldingUse : player.isHoldingUse;
        player.isHoldingSkill = playerPacket.isHoldingSkill != null ?
  playerPacket.isHoldingSkill : player.isHoldingSkill;
        players.put(player.id, player);
    }
}
```

```

public void addPlayer(MPPlayer player)
{
  this.players.put(player.c.getID(), player);
  this.network.sendGameBoard(gameBoard, player);

}

public void removePlayer(int id)
{

  players.remove(id);
}


  private class GameCycleThread extends Thread
  {
    volatile boolean isGameRunning = true;
    private final int gameSpeed = 16; //The lower the number the
 faster the game is

    public GameCycleThread()
    {
      this.isGameRunning = true;
    }

      public void run()
      {
        while (this.isGameRunning)
        {
            try
            {
                //Probably should use Timer instead
                Thread.sleep(gameSpeed);
            this.update();
        }
            catch (InterruptedException e)
            {
            e.printStackTrace();
            this.stopGame();
        }
        }
    }

    public void stopGame()
    {
        this.isGameRunning = false;

        //network should probably be closed by the parent
        network.close();
    }

    private void update()
    {
        updatePlayers();
        gameBoard.runTick();
    }

    private void updatePlayers()
    {
    for(MPPlayer p : players.values())
    {
      if (p.isHoldingPause)
    {

    }

      if (p.isHoldingSkill)
      {
        p.tryUsingSpell();
      }

      p.onTick();

    if (p.isHoldingUse)
    {
      gameBoard.SpawnBomb(p);
    }

    p.coordinate = checkCollision(p);

    network.sendGameBoard(gameBoard, p);
      network.sendPlayerInfo(p, true);
    }
    }

    private Vector2f checkCollision(MPPlayer p)
```

```
        {

          Vector2f coordsAfterMove = new Vector2f(p.coordinate.x, p.
coordinate.y);

          float padding = 0.001f;
      float cellSize = gameBoard.cellSize();

      boolean moveX = true;
      boolean moveY = true;

      if (p.isHoldingLeft)
      {
        coordsAfterMove.x -= p.speed;
      }

      if (p.isHoldingRight)
      {

        coordsAfterMove.x += p.speed;
      }

      boolean collidingLeft = ((int)coordsAfterMove.x / cellSize -
padding)<((int)p.coordinate.x / cellSize);
      boolean collidingRight=(((int)coordsAfterMove.x+p.size+
padding)/cellSize)>(((int)p.coordinate.x+p.size)/cellSize);
      boolean isCollidingX = collidingLeft || collidingRight;
      moveX = !(coordsAfterMove.x <= 0 || coordsAfterMove.x >=
gameBoard.size - p.size);
```

```
    //Some smoothing when going around edges would be nice
    if (isCollidingX && moveX)
    {
        int x = 0, y = 0, y1 = 0;
        if (collidingLeft)
        {
            x = (int) ((coordsAfterMove.x) / cellSize);
            y = (int) (p.coordinate.y / cellSize);
            y1 = (int) ((p.coordinate.y + p.size) / cellSize);

        }
        if (collidingRight)
        {
            x = (int) ((coordsAfterMove.x + p.size) / cellSize);
            y = (int) (p.coordinate.y / cellSize);
            y1 = (int) ((p.coordinate.y + p.size) / cellSize);

        }

        if (y == y1)
        {
            moveX = gameBoard.objects[x][y].isWalkable;
            gameBoard.objects[x][y].onStep(p);
        }
        else
        {
            moveX = gameBoard.objects[x][y].isWalkable && gameBoard.
objects[x][y1].isWalkable;
            gameBoard.objects[x][y].onStep(p);
            gameBoard.objects[x][y1].onStep(p);
        }
    }


    if (p.isHoldingUp)
    {
        coordsAfterMove.y += p.speed;
    }

    if (p.isHoldingDown)
    {
        coordsAfterMove.y -= p.speed;
    }

    boolean collidingUp = (((int)coordsAfterMove.y + p.size - padding
) / cellSize) > (((int)p.coordinate.y + p.size) / cellSize);
    boolean collidingDown = ((int)coordsAfterMove.y/cellSize+
padding)< ((int)p.coordinate.y/cellSize);
    boolean isCollidingY = collidingUp || collidingDown;
    moveY = !(coordsAfterMove.y <= 0 || coordsAfterMove.y >=
gameBoard.size - p.size);

    //Some smoothing when going around edges would be nice
    if (isCollidingY && moveY)
    {
        int x = 0, x1 = 0, y = 0;

        if (collidingUp)
        {
            y = (int) ((coordsAfterMove.y + p.size) / cellSize);
            x = (int) (p.coordinate.x / cellSize);
            x1 = (int) ((p.coordinate.x + p.size) / cellSize);

        }
        if (collidingDown)
        {
            y = (int) ((coordsAfterMove.y) / cellSize);
            x = (int) (p.coordinate.x / cellSize);
            x1 = (int) ((p.coordinate.x + p.size) / cellSize);
        }

        if (x == x1)
        {
            moveY = gameBoard.objects[x][y].isWalkable;
            gameBoard.objects[x][y].onStep(p);
        }
        else
        {
            moveY = gameBoard.objects[x][y].isWalkable && gameBoard.
    objects[x1][y].isWalkable;
            gameBoard.objects[x][y].onStep(p);
            gameBoard.objects[x1][y].onStep(p);
        }
    }

    coordsAfterMove.x = moveX ? coordsAfterMove.x : p.coordinate.x;
    coordsAfterMove.y = moveY ? coordsAfterMove.y : p.coordinate.y;
```

```

        return coordsAfterMove;

            }

        }

}
```

```
package server;

import org.lwjgl.system.CallbackI.P;

import com.esotericsoftware.kryonet.Connection;
import com.esotericsoftware.kryonet.Listener;
import com.esotericsoftware.kryonet.Server;

import shared.PacketUpdatePlayerPos;
import shared.SimplifiedGameBoard;
import shared.SimplifiedGameObject;
import shared.ObjectType;
import shared.PacketAddPlayer;
import shared.PacketRemovePlayer;
import shared.PacketUpdateGameBoard;
import shared.Vector2f;

public class Network extends Listener
{
  private Server server;
private final int port = 27960;

public boolean initKryoServer()
{
  try
  {
    this.server = new Server(131072, 16384);
    this.server.getKryo().register(PacketUpdatePlayerPos.class);
    this.server.getKryo().register(PacketAddPlayer.class);
    this.server.getKryo().register(PacketRemovePlayer.class);
    this.server.getKryo().register(PacketUpdateGameBoard.class);
    this.server.getKryo().register(SimplifiedGameBoard.class);
    this.server.getKryo().register(ObjectType.class);
    this.server.getKryo().register(SimplifiedGameObject.class);
    this.server.getKryo().register(SimplifiedGameObject[].class);
    this.server.getKryo().register(SimplifiedGameObject[][].class);
    this.server.getKryo().register(Vector2f.class);
    this.server.bind(this.port, this.port);
    this.server.addListener(this);
    this.server.start();
    System.out.println("The server is ready");
    return true;
  }
  catch (Exception e)
  {
    e.printStackTrace();
    return false;
  }
}
```

```

public void close()
{
    this.server.close();
}

public void connected(Connection c)
{
    MPPlayer player = new MPPlayer();
    player.c = c;

    PacketAddPlayer packet = new PacketAddPlayer();
    packet.id = c.getID();
    this.server.sendToAllExceptTCP(c.getID(), packet);
    GameServer.getInstance().addPlayer(player);
    System.out.println("Connection received.");
}

public void sendPlayerInfo(MPPlayer player, boolean isAccepted)
{
    PacketUpdatePlayerPos packet = new PacketUpdatePlayerPos(player.c.
  getID(), player);
    packet.accepted = isAccepted;
    this.server.sendToAllUDP(packet);

}

public void sendGameBoard(GameBoard gameBoard, MPPlayer player)
{
        SimplifiedGameBoard simpleGameboard = gameBoard.getSimpleGameBoard
      ();
        if (player == null)
        {
            PacketUpdateGameBoard packet = new PacketUpdateGameBoard(
      simpleGameboard);
            this.server.sendToAllUDP(packet);
        }
        else
        {
            PacketUpdateGameBoard packet = new PacketUpdateGameBoard(
      simpleGameboard);
            this.server.sendToUDP(player.c.getID(), packet);
        }
    }

    public void received(Connection connection, Object object)
    {
        if(object instanceof PacketUpdatePlayerPos)
        {
            PacketUpdatePlayerPos packet = (PacketUpdatePlayerPos) object;
            GameServer.getInstance().updatePlayer(connection.getID(), packet)
      ;
            System.out.println("Received coordinate packet");
        }
    }

    public void disconnected(Connection c)
    {
```

```java
        PacketRemovePlayer packet = new PacketRemovePlayer();
        packet.id = c.getID();
        this.server.sendToAllExceptTCP(c.getID(), packet);
        GameServer.getInstance().removePlayer(c.getID());
        System.out.println("Connection dropped.");
    }
}
```

```java
package server;

import java.util.ArrayList;
import java.util.Random;

import shared.*;

public class GameBoard implements Cloneable {

    public final int size = 1000;
    public final int gridSize = 20;
    public GameObject[][] objects;
    private BombObserver bombObserver;
    private AbstractFactory factory;

    private int currentTick = 0;
    private int powerUpCounter = 0;
    private int timeToCreatePowerUp = 10;

    private IStageBuilder stage1builder;
    private IStageBuilder stage2builder;
```

```java
  private IStageBuilder stage3builder;

  public GameBoard(AbstractFactory factory)
  {
      this.factory = factory;
      this.objects = new GameObject[gridSize][gridSize];
      this.stage1builder = new Stage1Builder(gridSize);
      this.stage2builder = new Stage2Builder(gridSize);
      this.stage3builder = new Stage3Builder(gridSize);

      StageDirector stageDirector = new StageDirector(stage3builder);

      stageDirector.makeStage();

      Stage stage1 = stageDirector.getStage();

      for(Coordinates kor: stage1.getGrounds())
          this.objects[kor.getX()][kor.getY()]=this.factory.
    createGround();

      int kiekis = 76;
      boolean sunaikinama = false;

      for(Coordinates kor: stage1.getWalls()){

          this.objects[kor.getX()][kor.getY()]=this.factory.
    createWall(sunaikinama);
          kiekis -=1;
          if(kiekis == 0) sunaikinama = true;
```

```
        }




        bombObserver = new BombObserver(this);


        this.factory.SetBombObserver(bombObserver);






        //test

        //TODO wrong but ok for now
        this.objects[17][3] = this.factory.createTrap();

        this.objects[15][3] = this.factory.createTrap();
        Trap modified = (Trap) this.objects[15][3];
        modified.setTrapeffect(new DamageTrap (new ConcreteTrap()));
        this.objects[15][3] = modified;

        this.objects[13][3] = this.factory.createTrap();
        Trap modified1 = (Trap) this.objects[13][3];
        modified1.setTrapeffect(new TeleportTrap (new ConcreteTrap()));
        this.objects[13][3] = modified1;

        this.objects[11][3] = this.factory.createTrap();
        Trap modified2 = (Trap) this.objects[11][3];
        modified2.setTrapeffect(new SlowTrap (new ConcreteTrap()));
        this.objects[11][3] = modified2;

        this.objects[9][3] = this.factory.createTrap();
        Trap modified3 = (Trap) this.objects[9][3];
        modified3.setTrapeffect(new DamageTrap(new TeleportTrap (new
    SlowTrap (new ConcreteTrap()))));
        this.objects[9][3] = modified3;

    }

    public void paleistiKopija(){
        GameBoard copy = copyDeep();


    }



    public GameBoard copyDeep(){

        try {
            return (GameBoard) this.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
            return null;
        }
    }
```

```

  public void SpawnBomb(PlayerInfo player)
  {
      int x = Math.round(player.coordinate.x/(size/gridSize));
      int y = Math.round(player.coordinate.y/(size/gridSize));
      this.objects[x][y] = this.factory.createBomb(player.id);

  }

  public void ClearTarget(int x, int y){
      this.objects[x][y] = this.factory.createGround();
      System.out.println("Removing bomb from location " + x + " " + y
);
  }

  public SimplifiedGameBoard getSimpleGameBoard()
  {
  SimplifiedGameBoard simpleGameboard = new SimplifiedGameBoard(this.
size, this.gridSize);

      for (int i = 0; i < this.gridSize; i ++)
      {
          for (int j = 0; j < this.gridSize; j ++)
          {
             simpleGameboard.objects[i][j] = new SimplifiedGameObject
(this.objects[i][j].color, ObjectType.GROUND);
          }
      }
      return simpleGameboard;
```

```
    }

    private void spawnPowerUp(){
        Random rand = new Random();

        if(powerUpCounter > 4)
            return;

        while(true){
            int x = rand.nextInt((19 - 0) + 1) + 0;
            int y = rand.nextInt((19 - 0) + 1) + 0;


            if(this.objects[x][y] instanceof Ground){
                this.objects[x][y] = this.factory.createPowerUp();
                powerUpCounter+=1;
                break;
            }

        }









    }
```

```java


    public void runTick()
    {
        currentTick +=1;

        if(currentTick % (60 * timeToCreatePowerUp)== 0) // kas 10s sukurti nauja powerUp
            spawnPowerUp();

        for(int x = 0; x < this.gridSize; x++)
        {
            for (int y = 0; y < this.gridSize; y++)
            {
              if(this.objects[x][y].isDead) {



                    if(this.objects[x][y] instanceof PowerUp)
                    {
                        System.out.println("Sunaikinau powerUpa");
                        powerUpCounter -=1;
                    }

                    this.ClearTarget(x, y);



                }
```

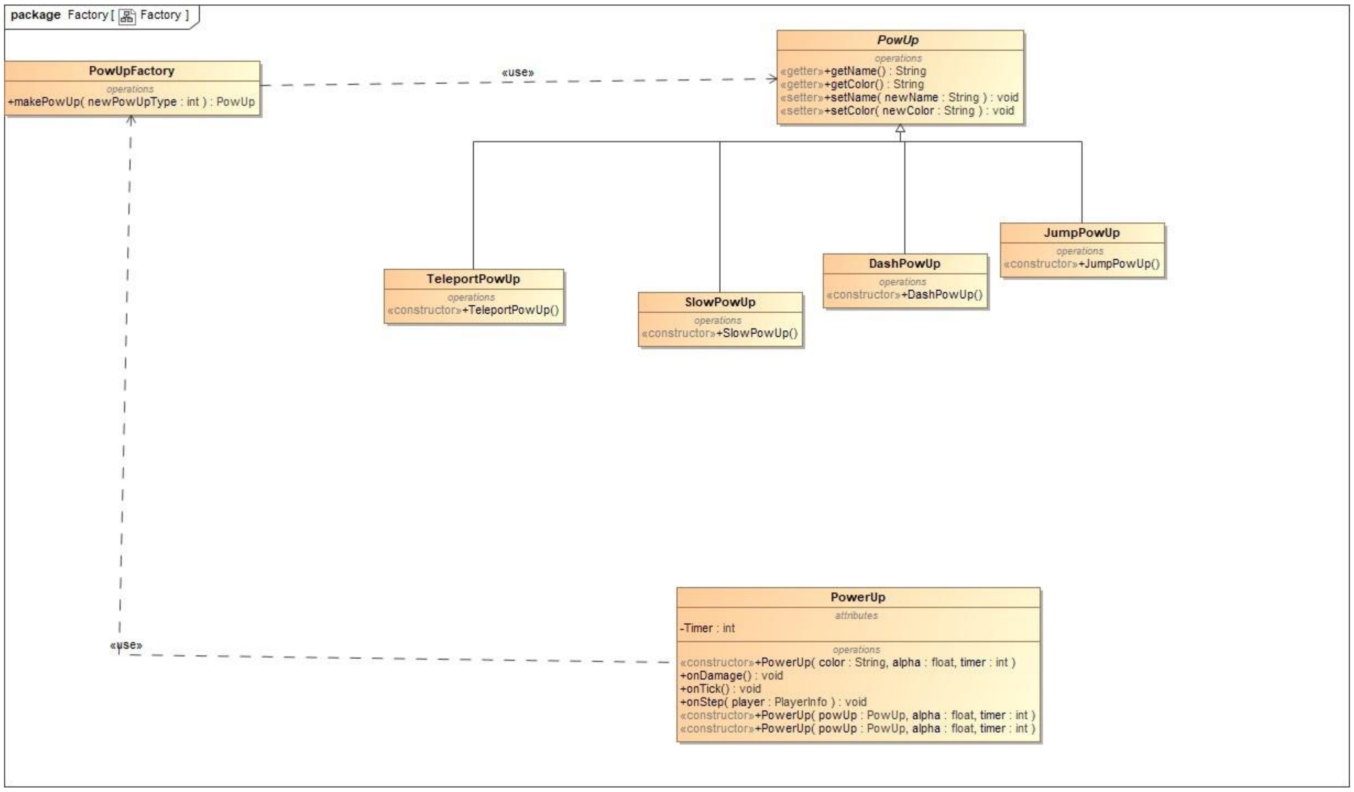
```

                    objects[x][y].onTick();
                }
            }
        }

        public int cellSize()
        {
            return size / gridSize;
        }
}
```

```
package server;

import shared.Vector2f;

public class PlayerInfo
{
    public int id;
    public Vector2f coordinate;
    public boolean isHoldingLeft;
    public boolean isHoldingRight;
    public boolean isHoldingUp;
    public boolean isHoldingDown;
    public boolean isHoldingUse;
    public boolean isHoldingSkill;
    public boolean isHoldingPause;
    public boolean isHoldingUnPause;
    public boolean placedBomb = true;
public int health;
public float speed;
public float baseSpeed;
public int size;
public int bombCount;

private SkillAlgorithm skillAlgorithm;

public PlayerInfo()
{
  this.coordinate = new Vector2f();
  this.coordinate.x = 400;
  this.coordinate.y = 400;
  this.isHoldingLeft = false;
  this.isHoldingRight = false;
  this.isHoldingUp = false;
  this.isHoldingDown = false;
  this.isHoldingPause = false;
  this.isHoldingUnPause = false;
  this.skillAlgorithm = new DashSkill();
  this.size = 40;
  this.speed = 2.5f;
  this.baseSpeed = 2.5f;
  this.health = 3;
  this.bombCount = 2;
  //this.playerStats = new ConcretePlayer();
}

public PlayerInfo(int id, Vector2f coordinate)
```

```
{
    this.id = id;
    this.coordinate = coordinate;
    this.isHoldingLeft = false;
    this.isHoldingRight = false;
    this.isHoldingUp = false;
    this.isHoldingDown = false;
    this.isHoldingUse = false;
    this.isHoldingPause = false;
    this.isHoldingUnPause = false;
    this.size = 40;
    this.speed = 2.5f;
    this.baseSpeed = 2.5f;
    this.health = 3;
    this.bombCount = 2;
    //Test
}

public SkillAlgorithm getSkillAlgorithm()
{
    return skillAlgorithm;
}

public void setSkillAlgorithm(SkillAlgorithm skillAlgorithm)
{
    this.skillAlgorithm = skillAlgorithm;
}


```

```
  public void onTick()
  {
    this.skillAlgorithm.onTick(this);
  }

  public void tryUsingSpell()
  {
    this.skillAlgorithm.useSkill(this);
  }

  public int getCooldown()
  {
    return this.skillAlgorithm.getCooldown();
  }

  public String getName()
  {
    return this.skillAlgorithm.getName();
  }
}
```

5.1.7. Factory


package Factory [Factory]
PowUpFactory
operations
+makePowUp(newPowUpType : int) : PowUp
«use»
PowUp
operations
«getter»+getName() : String
«getter»+getColor() : String
«setter»+setName(newName : String) : void
«setter»+setColor(newColor : String) : void
TeleportPowUp
operations
«constructor»+TeleportPowUp()
SlowPowUp
operations
«constructor»+SlowPowUp()
DashPowUp
operations
«constructor»+DashPowUp()
JumpPowUp
operations
«constructor»+JumpPowUp()
«use»
PowerUp
attributes
-Timer : int
operations
«constructor»+PowerUp(color : String, alpha : float, timer : int)
+onDamage() : void
+onTick() : void
+onStep(player : PlayerInfo) : void
«constructor»+PowerUp(powUp : PowUp, alpha : float, timer : int)
«constructor»+PowerUp(powUp : PowUp, alpha : float, timer : int)

5.11 pav. Factory diagrama

Factory šablono poreikis atsirado norint kad dinamiškai keistusį žaidėjo įgudžiai.

```
package server;

public class PowUpFactory {
    public PowUp makePowUp(int newPowUpType) {
        PowUp newPowUp = null;

        if(1 == newPowUpType)
            return new JumpPowUp();

        else if(2 == newPowUpType)
            return new DashPowUp();

        else if(3 == newPowUpType)
            return new SlowPowUp();

        else if(4 == newPowUpType)
            return new TeleportPowUp();

        return newPowUp;
    }


}
```

```
package server;

public abstract class PowUp {

    private String name;
    private String color;



    public String getName(){
        return name;
    }

    public String getColor(){
        return color;
    }
```

```java

    public void setName(String newName){
        name = newName;
    }

    public void setColor(String newColor){
        color = newColor;
    }


}
```

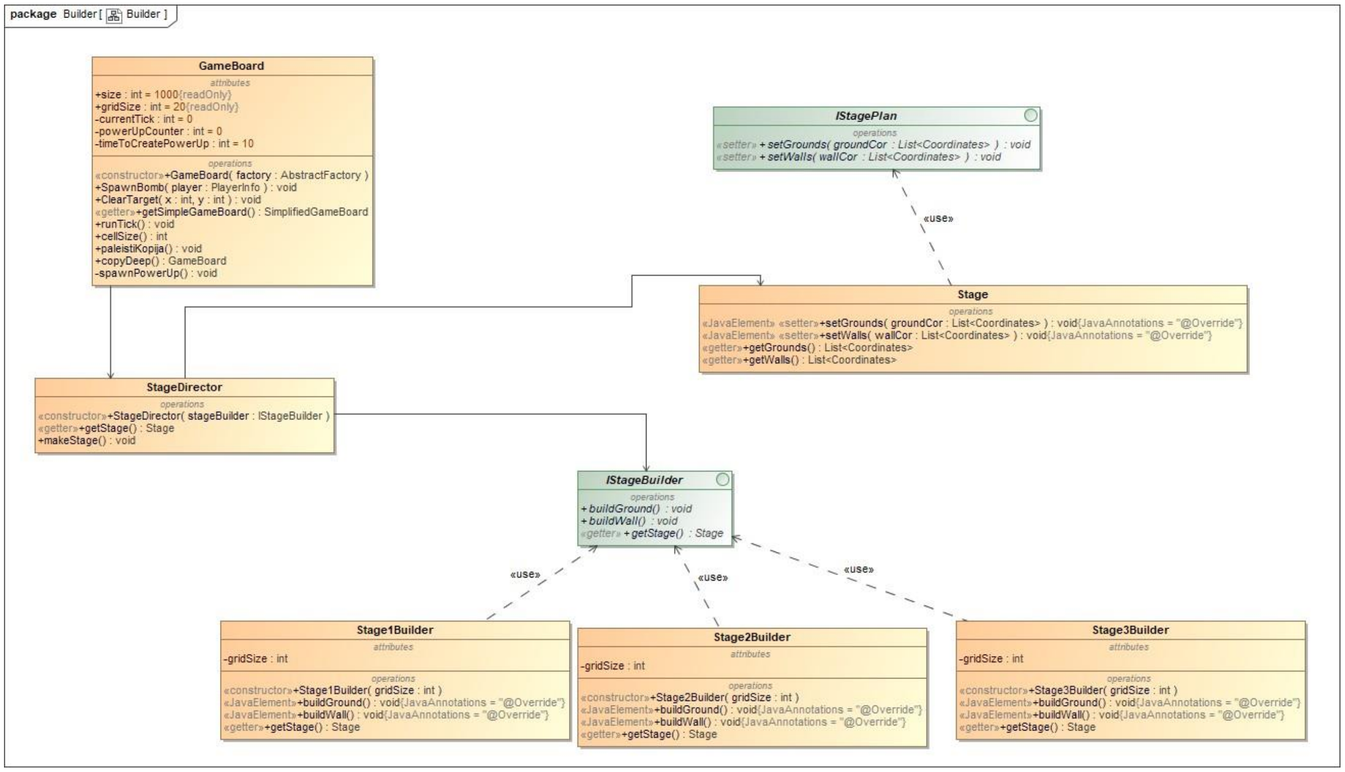
```java
package server;

public class SlowPowUp extends PowUp{
    public SlowPowUp(){
        setName("Slow");
        setColor("#0728ab");
    }
}
```

```java
package server;

public class DashPowUp extends PowUp{

    public DashPowUp(){
        setName("Dash");
        setColor("#ccbb23");
    }
}
```

```java
package server;

public class JumpPowUp extends PowUp{
    public JumpPowUp(){
        setName("Jump");
        setColor("#ab4907");
    }
}
```

```java
package server;

public class TeleportPowUp extends PowUp{

    public TeleportPowUp(){
        setName("Teleport");
        setColor("#7b219e");
    }
}
```

5.1.8. Builder


package Builder[Builder]
GameBoard
attributes
+size : int = 1000{readOnly}
+gridSize : int = 20{readOnly}
-currentTick : int = 0
-powerUpCounter : int = 0
-timeToCreatePowerUp : int = 10
operations
«constructor»+GameBoard(factory : AbstractFactory)
+SpawnBomb(player : PlayerInfo) : void
+ClearTarget(x : int, y : int) : void
«getter»+getSimpleGameBoard() : SimplifiedGameBoard
+runTick() : void
+cellSize() : int
+paleistiKopija() : void
+copyDeep() : GameBoard
-spawnPowerUp() : void
IStagePlan
operations
«setter» + setGrounds(groundCor : List<Coordinates>) : void
«setter» + setWalls(wallCor : List<Coordinates>) : void
«use»
Stage
operations
«JavaElement» «setter»+setGrounds(groundCor : List<Coordinates>) : void{JavaAnnotations = "@Override"}
«JavaElement» «setter»+setWalls(wallCor : List<Coordinates>) : void{JavaAnnotations = "@Override"}
«getter»+getGrounds() : List<Coordinates>
«getter»+getWalls() : List<Coordinates>
StageDirector
operations
«constructor»+StageDirector(stageBuilder : IStageBuilder)
«getter»+getStage() : Stage
+makeStage() : void
IStageBuilder
operations
+ buildGround() : void
+ buildWall() : void
«getter» + getStage() : Stage
«use»
«use»
«use»
Stage1Builder
attributes
-gridSize : int
operations
«constructor»+Stage1Builder(gridSize : int)
«JavaElement»+buildGround() : void{JavaAnnotations = "@Override"}
«JavaElement»+buildWall() : void{JavaAnnotations = "@Override"}
«getter»+getStage() : Stage
Stage2Builder
attributes
-gridSize : int
operations
«constructor»+Stage2Builder(gridSize : int)
«JavaElement»+buildGround() : void{JavaAnnotations = "@Override"}
«JavaElement»+buildWall() : void{JavaAnnotations = "@Override"}
«getter»+getStage() : Stage
Stage3Builder
attributes
-gridSize : int
operations
«constructor»+Stage3Builder(gridSize : int)
«JavaElement»+buildGround() : void{JavaAnnotations = "@Override"}
«JavaElement»+buildWall() : void{JavaAnnotations = "@Override"}
«getter»+getStage() : Stage

5.12 pav. Builder diagrama

Builder šablono pereikis atsirado norint palengvinti kiekvieno žaidimo lygio kurimą.

```java
package server;

import java.util.List;

public class Stage implements IStagePlan{


    private List <Coordinates> grounCor;
    private List<Coordinates> wallCor;

    @Override
    public void setGrounds(List<Coordinates> groundCor) {
        this.grounCor = groundCor;
    }

    @Override
    public void setWalls(List<Coordinates> wallCor) {
        this.wallCor = wallCor;
    }

    public List<Coordinates> getGrounds(){
        return this.grounCor;
    }

    public List<Coordinates> getWalls(){
        return this.wallCor;
    }









}
```

```java
package server;

public class StageDirector {
    private IStageBuilder stageBuilder;

    public StageDirector(IStageBuilder stageBuilder){
        this.stageBuilder = stageBuilder;
    }

    public Stage getStage(){
        return this.stageBuilder.getStage();
    }


    public void makeStage(){
        this.stageBuilder.buildWall();
        this.stageBuilder.buildGround();
    }
}
```

```java
package server;

import java.util.ArrayList;
import java.util.List;

public class Stage1Builder implements IStageBuilder{
    private Stage stage;
    private int gridSize;

    public Stage1Builder(int gridSize){
        this.stage = new Stage();
        this.gridSize = gridSize;
    }

    @Override
    public void buildGround() {
        List<Coordinates> ground = new ArrayList<Coordinates>();

        for(int x = 0; x<gridSize; x++){
            for(int y = 0; y<gridSize; y++){
                if(x != 0 && x != gridSize-1 && y != 0 && y != gridSize
    -1)
                    ground.add(new Coordinates(x, y));
            }
        }


        stage.setGrounds(ground);
    }

    @Override
    public void buildWall() {
        List<Coordinates> walls = new ArrayList<Coordinates>();

        for(int x = 0; x<gridSize; x++){
            for(int y = 0; y<gridSize; y++){
                if(x == 0 || x == gridSize-1 || y == 0 || y == gridSize
    -1)
                    walls.add(new Coordinates(x, y));
            }
        }
```

```java

        System.out.println("Koks dydis Pries: " + walls.size());

        walls.add(new Coordinates(5, 4));
        walls.add(new Coordinates(5, 6));
        walls.add(new Coordinates(6, 5));

        walls.add(new Coordinates(15, 16));
        walls.add(new Coordinates(15, 14));
        walls.add(new Coordinates(14, 15));

        for(int x = 1; x < 19; x++){
            walls.add(new Coordinates(x, 11));
        }

        System.out.println("Koks dydis Po: " + walls.size());

        stage.setWalls(walls);
    }

    public Stage getStage(){
        return this.stage;
    }

}
```

```java
package server;

import java.util.ArrayList;
import java.util.List;

public class Stage2Builder implements IStageBuilder{
    private Stage stage;
    private int gridSize;

    public Stage2Builder(int gridSize){
        this.stage = new Stage();
        this.gridSize = gridSize;
    }

    @Override
    public void buildGround() {
        List<Coordinates> ground = new ArrayList<Coordinates>();

        for(int x = 0; x<gridSize; x++){
            for(int y = 0; y<gridSize; y++){
                if(x != 0 && x != gridSize-1 && y != 0 && y != gridSize
    -1)
                    ground.add(new Coordinates(x, y));
            }
        }


        stage.setGrounds(ground);
    }

    @Override
    public void buildWall() {
        List<Coordinates> walls = new ArrayList<Coordinates>();

        for(int x = 0; x<gridSize; x++){
            for(int y = 0; y<gridSize; y++){
                if(x == 0 || x == gridSize-1 || y == 0 || y == gridSize
    -1)
                    walls.add(new Coordinates(x, y));
            }
        }

        System.out.println("Koks dydis Pries: " + walls.size());




        for(int i = 7; i<11; i++){
            walls.add(new Coordinates(i, 9));
            walls.add(new Coordinates(i, 13));
            walls.add(new Coordinates(i, 17));
        }

        for(int i =10; i<17; i++){
            if(i != 13){
                walls.add(new Coordinates(6, i));
                walls.add(new Coordinates(11, i));
            }
        }






        System.out.println("Koks dydis Po:  " + walls.size());

        stage.setWalls(walls);
    }

    public Stage getStage(){
        return this.stage;
    }

}
```

```java
package server;

import java.util.ArrayList;
import java.util.List;

public class Stage3Builder implements IStageBuilder{
    private Stage stage;
    private int gridSize;

    public Stage3Builder(int gridSize){
        this.stage = new Stage();
        this.gridSize = gridSize;
    }

```

```
@Override
public void buildGround() {
    List<Coordinates> ground = new ArrayList<Coordinates>();

    for(int x = 0; x<gridSize; x++){
        for(int y = 0; y<gridSize; y++){
            if(x != 0 && x != gridSize-1 && y != 0 && y != gridSize
-1)
                ground.add(new Coordinates(x, y));
        }
    }


    stage.setGrounds(ground);
}

@Override
public void buildWall() {
    List<Coordinates> walls = new ArrayList<Coordinates>();

    for(int x = 0; x<gridSize; x++){
        for(int y = 0; y<gridSize; y++){
            if(x == 0 || x == gridSize-1 || y == 0 || y == gridSize
-1)
                walls.add(new Coordinates(x, y));
        }
    }

    System.out.println("Koks dydis Pries: " + walls.size());
```

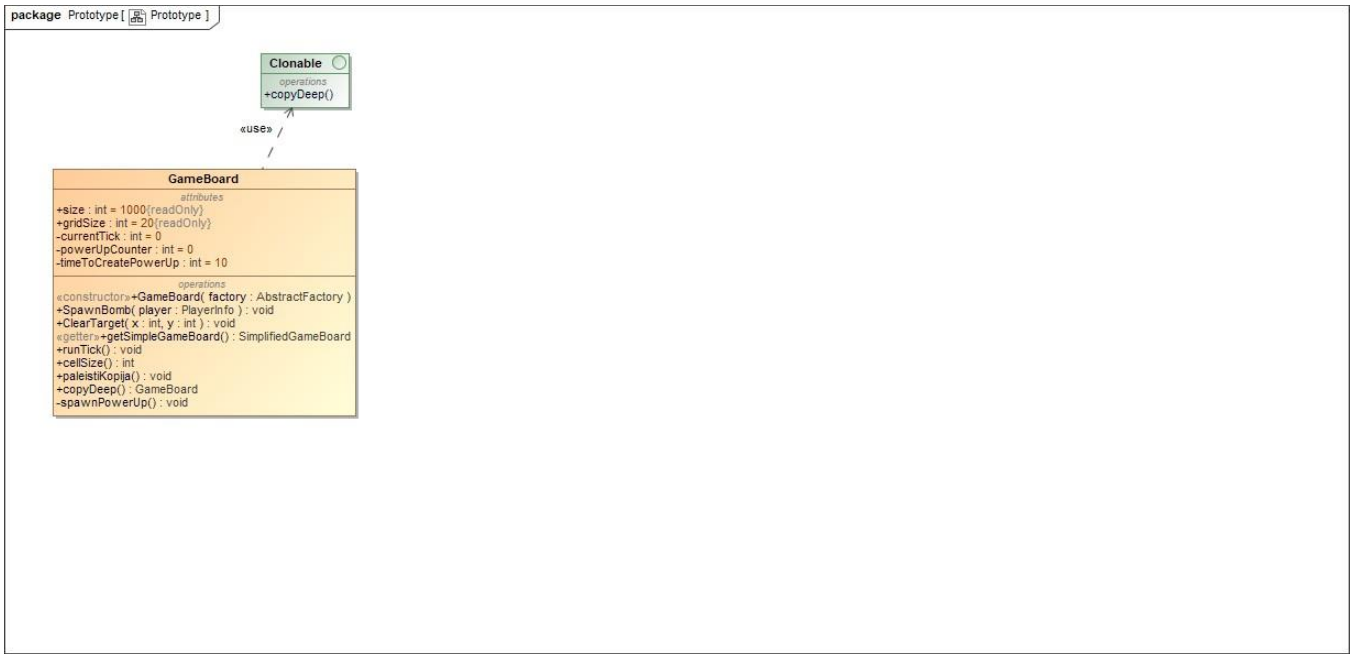
```


        walls.add(new Coordinates(3, 14-1));
        walls.add(new Coordinates(4, 14-1));
        walls.add(new Coordinates(5, 14-1));
        walls.add(new Coordinates(6, 14-1));
        walls.add(new Coordinates(3, 18-1));
        walls.add(new Coordinates(4, 18-1));
        walls.add(new Coordinates(5, 18-1));
        walls.add(new Coordinates(6, 18-1));
        walls.add(new Coordinates(3, 17-1));
        walls.add(new Coordinates(3, 16-1));
        walls.add(new Coordinates(3, 15-1));
        walls.add(new Coordinates(5, 16-1));
        walls.add(new Coordinates(6, 16-1));
        walls.add(new Coordinates(6, 15-1));





        walls.add(new Coordinates(8, 14-1));
        walls.add(new Coordinates(8, 15-1));
        walls.add(new Coordinates(8, 16-1));
        walls.add(new Coordinates(8, 17-1));
        walls.add(new Coordinates(11, 14-1));
        walls.add(new Coordinates(11, 15-1));
        walls.add(new Coordinates(11, 16-1));
        walls.add(new Coordinates(11, 17-1));
        walls.add(new Coordinates(9, 16-1));
        walls.add(new Coordinates(10, 16-1));
        walls.add(new Coordinates(9, 18-1));
        walls.add(new Coordinates(10, 18-1));



        walls.add(new Coordinates(13, 14-1));
        walls.add(new Coordinates(14, 14-1));
        walls.add(new Coordinates(15, 14-1));
        walls.add(new Coordinates(16, 14-1));

        walls.add(new Coordinates(13, 15-1));
        walls.add(new Coordinates(13, 16-1));
        walls.add(new Coordinates(13, 17-1));
        walls.add(new Coordinates(13, 18-1));


        for(int i = 8; i<12; i++){
            walls.add(new Coordinates(i, 11-1));
            walls.add(new Coordinates(i, 8-1));
            walls.add(new Coordinates(i, 5-1));
        }

        walls.add(new Coordinates(8, 10-1));
        walls.add(new Coordinates(8, 9-1));
        walls.add(new Coordinates(11, 10-1));
        walls.add(new Coordinates(11, 9-1));
        walls.add(new Coordinates(11, 7-1));
```

```java
            walls.add(new Coordinates(11, 6-1));


            System.out.println("Koks dydis Po: " + walls.size());

            stage.setWalls(walls);
        }

        public Stage getStage(){
            return this.stage;
        }


}
```

```java
package server;

public interface IStageBuilder {
        public void buildGround();
        public void buildWall();

        public Stage getStage();
}
```

```java
package server;

import java.util.List;

public interface IStagePlan {

    public void setGrounds(List<Coordinates> groundCor);

    public void setWalls(List<Coordinates> wallCor);

}
```

5.1.9. Prototype



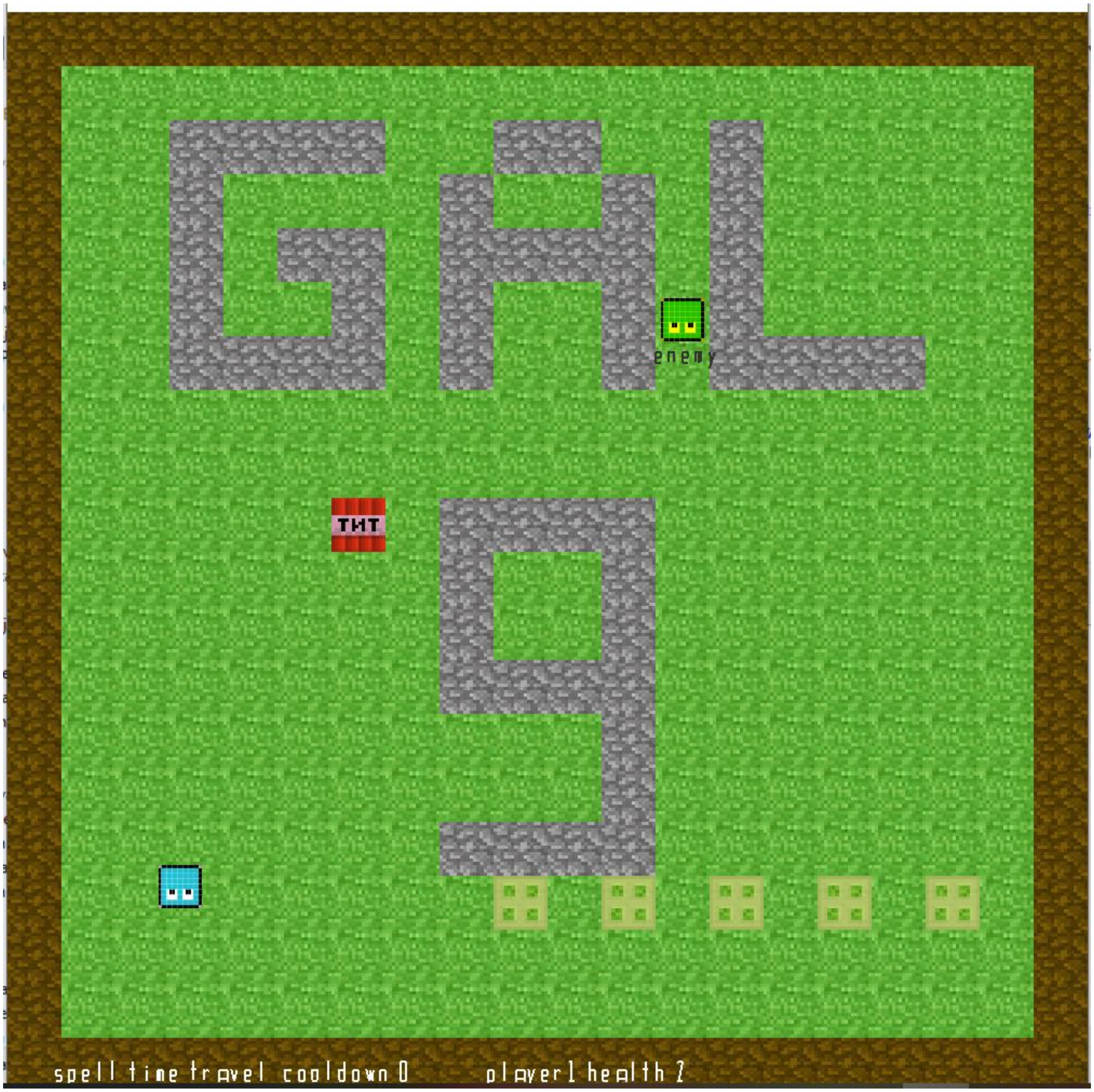
package Prototype [Prototype]
Clonable
operations
+copyDeep()
«use»
GameBoard
attributes
+size : int = 1000{readOnly}
+gridSize : int = 20{readOnly}
-currentTick : int = 0
-powerUpCounter : int = 0
-timeToCreatePowerUp : int = 10
operations
«constructor»+GameBoard(factory : AbstractFactory)
+SpawnBomb(player : PlayerInfo) : void
+ClearTarget(x : int, y : int) : void
«getter»+getSimpleGameBoard() : SimplifiedGameBoard
+runTick() : void
+cellSize() : int
+paleistiKopija() : void
+copyDeep() : GameBoard
-spawnPowerUp() : void

5.13 pav. Prototype diagrama

Prototype šablono poreikis atsirado norint kad butu galymi gryžtai jau ankščiau išsaugotą žaidimo stadiją, kad būtų galimą į ją grižti.

6. PROJEKTO APRAŠYMAS ANTRA DALIS

Žaidimas, kuriame strategiškai dėlioji bombas, kuriomis gali sunaikinti kliūtis bei priešus, išvengi-nėji spąstų bei naudoji įgytas galias kad įgautum pranašumą prieš savo varžovą.



6.1 pav. Žaidimo prototipo nuotrauka

7. ŽAIDIMO REIKALAVIMAI

ŽAIDIMO LYGIAI

Žaidimas susidės iš trijų lygių, kurie vienas nuo kito skirsis savo žaidimo strategijomis bei sudėtingumu.

PIRMASIS LYGIS

Jame bus sukurtas pasaulis, kuris susidės iš sienų, kurių kiaurai pereiti negalima, bet galima jas susprogdinti naudojant bombas, bei žemės, per kurią žaidėjas gales laisvai vaikščioti. Žaidimo tiklsas nugalėti savo priešininką, taktiškai naikinant sienas. Žaidėjai turės po 3 gyvybes, viena gyvybė yra prarandama jeigu savo arba priešininko bomba sprogsta šalia.

ANTRAS LYGIS

Antrame lygyje atsiranda nauja kliūtis- tai spąstai, kurie sugeneruojami sukuriant 2 lygio pasaulį. Visi spąstai atrodo taip pat tik žaidėjai nežino, ką jis gali padaryti, tik žino tą kad visi spąstai jį užsaldys. Taip pat spąstai gali žaidėja sulėtinti, nuimti gyvybę ar jį kažkur nukelti.

TREČIAS LYGIS

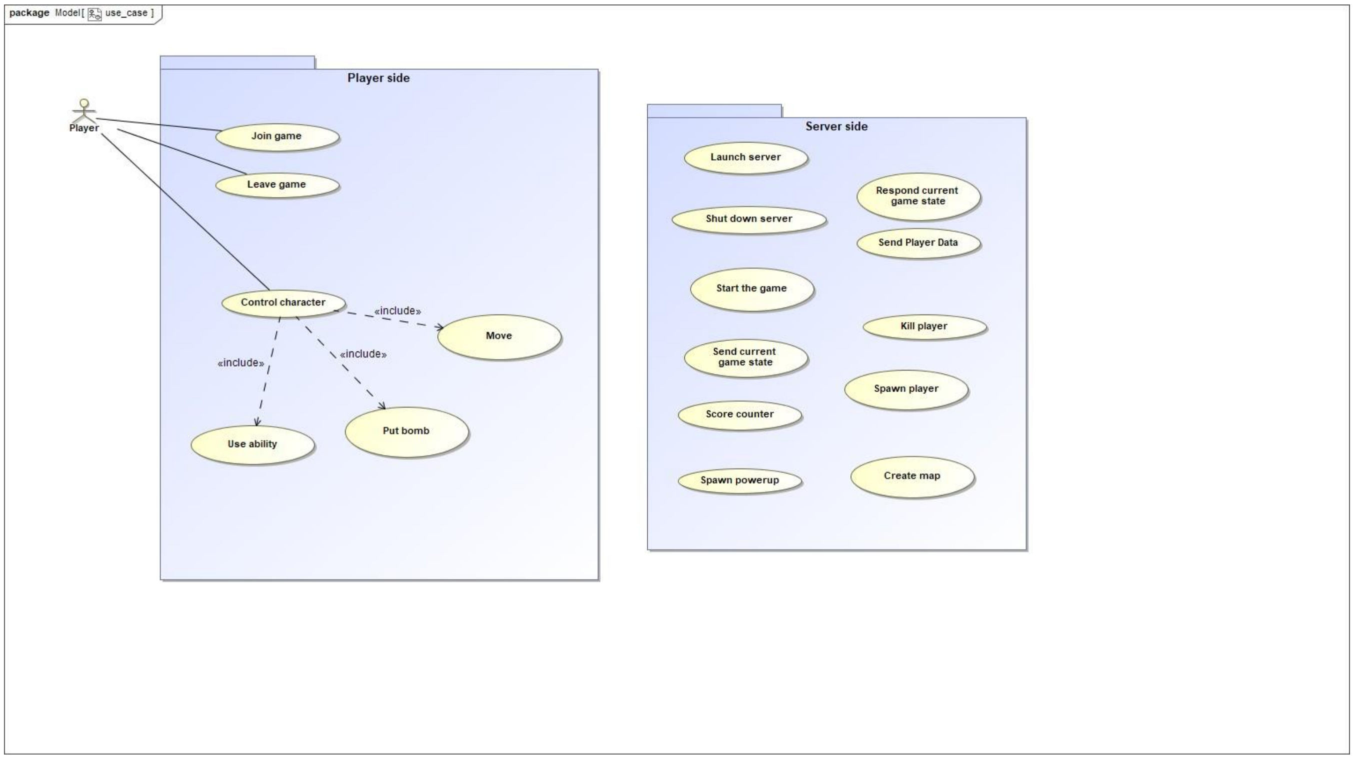
Trečiame lygyje atsiranda žaidėjo įgudis. Kai 3 lygio pasaulis yra sugeneruotas kas 10 sekundžių atsiranda žaidėjo įgudis, ant kurio užlipus žaidėjas gauna viena iš 4 pagerinimų:

- Sulėtinimas priešininko. Galima naudoti kas 30 sekundžių.
- Nusikelimas į naują vietą. Galima naudoti kas 20 sekundžių.
- Sienų peršokimas. Galima naudoti kas 15 sekundžių.
- Greitas žaidėjo paslinkimas. Galima naudoti kas 15 sekundžių.

8. PROJEKTUI NAUDOTOS TECHNOLOGIJOS

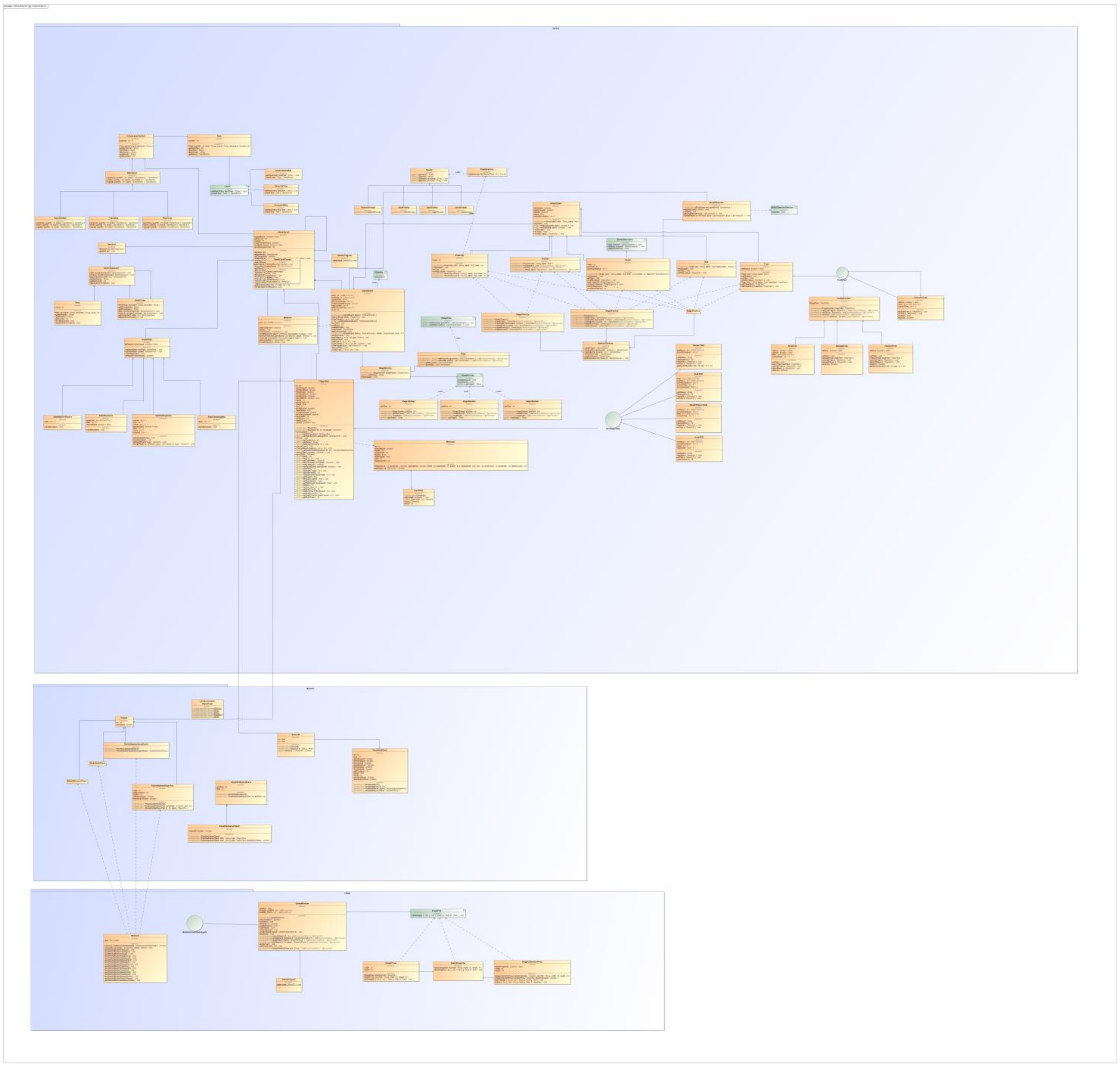
Serverio ir kliento komunikacijai naudotas Kryonet. Programos lango sukūrimui ir piešimui buvo pasitelkta lwjgl ir OpenGL 1.1 įrankiai.

9. USE CASE DIAGRAMA



4.2 pav. Use case diagrama

10. KLASIŲ DIAGRAMA



5.14 pav. Klasių diagrama

11 ŠABLONAI

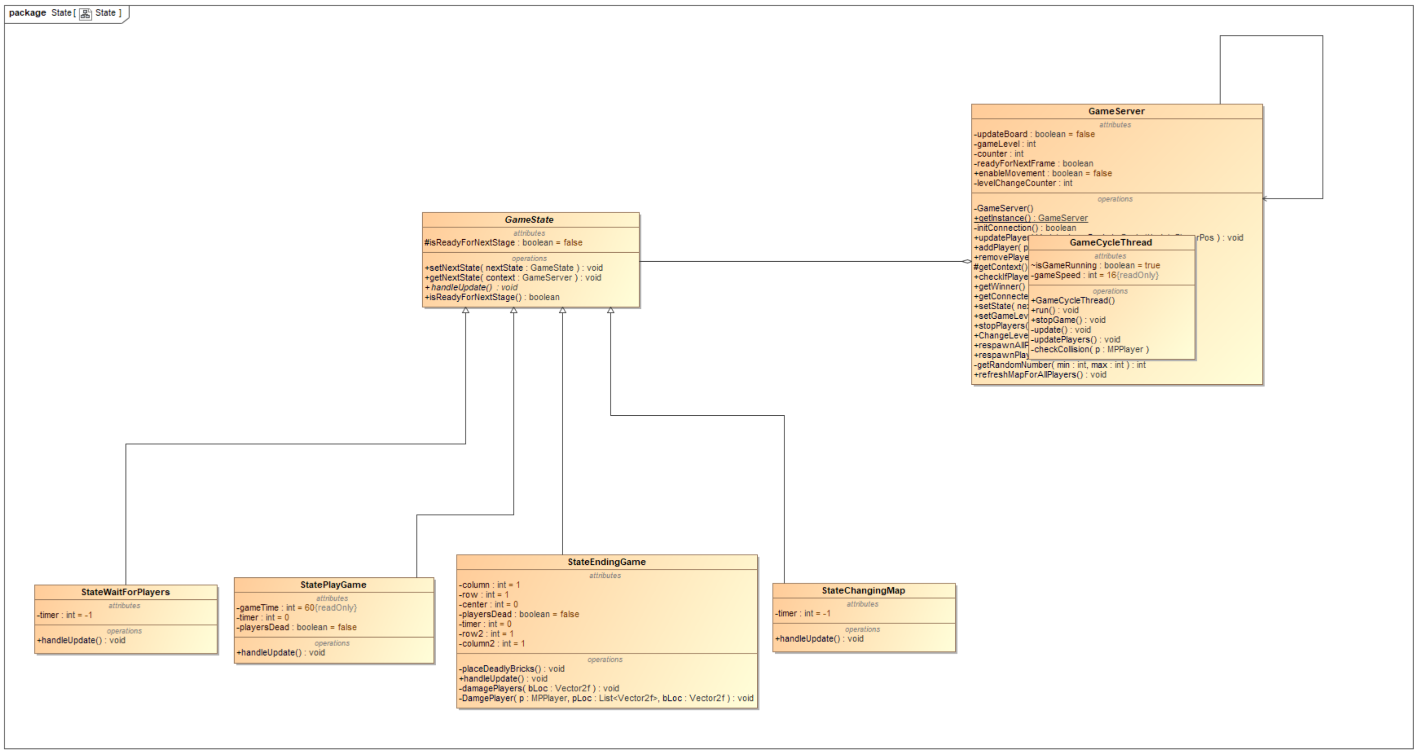
GameState

GameState šablono prireikė norint įdėti į žaidimą skirtingas būsenas. Tai yra :

- Žaidimo laukimas
- Žaidimo vykdymas
- Žaidimo užbaigimo bandymas
- Žaidimo lentos pakeitimas į kitą

Šis šablonas puikiai padėjo įgyvendinti šią idėją.

UML diagrama:



Kodas:

```java
package server;

public abstract class GameState
{
    private GameState nextState;
    protected boolean isReadyForNextStage = false;

    public void setNextState(GameState nextState) {
      this.nextState = nextState;
    }

     public void getNextState( GameServer context )
    {
      context.setState(nextState);
    }

     public abstract void handleUpdate();

     public boolean isReadyForNextStage()
    {
      return isReadyForNextStage;
    }


}

```

```java
package server;

```

```java
public class StatePlayGame extends GameState
{
  private final int gameTime = 60; // 1 minute
  private int timer = 0;
  private boolean playersDead = false;

  @Override
  public void handleUpdate()
  {
    GameServer gameServer = GameServer.getInstance();
    this.isReadyForNextStage = false;
    if (timer > gameTime * 60 || playersDead)
    {
      this.isReadyForNextStage = true;
      this.timer = 0;
      this.playersDead = false;
    }
    else
    {
      if (gameServer.checkIfPlayerDead())
      {
        this.playersDead = true;
        gameServer.enableMovement = false;
      }
      gameServer.enableMovement = true;
      this.timer ++;
    }

  }

}
```

```
package server;

public class StateChangingMap extends GameState
{
   private int timer = -1;

   @Override
   public void handleUpdate()
   {
      GameServer gameServer = GameServer.getInstance();
      gameServer.enableMovement = false;
      this.isReadyForNextStage = false;
      if (this.timer == -1)
      {
         this.timer = 300;
      }
      gameServer.network.sendString("Winner player: "+gameServer.
    getWinner() + " map refresh in... "+(this.timer / 60+1));
      if (timer == 0)
      {
         gameServer.network.sendString(null);
         this.isReadyForNextStage = true;
         this.timer = -1;
         gameServer.setGameLevel();
      }
      else
    {
      this.timer--;
    }

  }

}
```

```
package server;

public class StateWaitForPlayers extends GameState
{
  private int timer = -1;

  @Override
  public void handleUpdate()
  {
    GameServer gameServer = GameServer.getInstance();
    this.isReadyForNextStage = false;
    if (gameServer.getConnectedPlayerCount() > 2)
    {
      if (this.timer < 0)
      {
        this.timer = 300;
      }
      gameServer.network.sendString("Game is starting in.. " + (this.
   timer / 60 + 1));
      if (timer == 0)
      {
```

```java
                gameServer.network.sendString(null);
                this.isReadyForNextStage = true;
                this.timer = -1;
            }
            else
            {
                this.timer--;
            }
        }
        else
        {
            this.timer = -1;
            gameServer.network.sendString("Waiting for players..");
        }

    }

}
```

```java
package server;

import java.util.ArrayList;
import java.util.List;

import shared.Vector2f;

public class StateEndingGame extends GameState
{
    private int timer = 0;
    private boolean playersDead = false;
    private int row = 1;
    private int column = 1;
    private int row2 = 1;
    private int column2 = 1;
    private int center = 0;
    @Override
    public void handleUpdate()
    {
       GameServer gameServer = GameServer.getInstance();
       this.isReadyForNextStage = false;
       if (playersDead)
       {
          this.isReadyForNextStage = true;
          this.timer = 0;
          this.playersDead = false;
          row = 0;
          column = 1;
          row2 = 1;
          column2 = 1;
          center = 0;
       }
       else
       {
          timer ++;
          if (gameServer.checkIfPlayerDead())
          {
             this.playersDead = true;
             gameServer.enableMovement = false;
```

```
        }
        gameServer.enableMovement = true;
        placeDeadlyBricks();
    }

}

private void placeDeadlyBricks()
{
    GameServer gameServer = GameServer.getInstance();
    if (timer % 15 == 0)
    {
        if (row < gameServer.gameBoard.gridSize - 2 - center)
        {
            gameServer.gameBoard.addObject(row, column, "wall");
            this.damagePlayers(new Vector2f(row, column));
            row ++;
        }
        else if(column < gameServer.gameBoard.gridSize - 1 - center)
        {
            gameServer.gameBoard.addObject(row, column, "wall");
            this.damagePlayers(new Vector2f(row, column));
            column ++;
        }
        else if (row2 < gameServer.gameBoard.gridSize - 2 - center)
        {
            gameServer.gameBoard.addObject(gameServer.gameBoard.gridSize -
  row2 -2, column -1,"wall");
            this.damagePlayers(new Vector2f(gameServer.gameBoard.gridSize -
    row2  - 2,  column  -  1));
        row2 ++;
      }
      else  if(column2 < gameServer.gameBoard.gridSize  - 4  -  center)
      {
        gameServer.gameBoard.addObject(gameServer.gameBoard.gridSize  -
   row2 -1,  gameServer.gameBoard.gridSize  - column2 - 2,"wall");
        this.damagePlayers(new Vector2f(gameServer.gameBoard.gridSize  -
    row2 -1, gameServer.gameBoard.gridSize  - column2  - 2));
        column2 ++;
      }
      else
      {
        center ++;
        row = center;
        column = center + 1;
        row2 = center + 1;
        column2 = center + 1;
      }
    }
  }

    private void damagePlayers(Vector2f bLoc){

        GameServer session = GameServer.getInstance();

        int cellSize = session.gameBoard.cellSize();

        System.out.println("Kreipiuosi");


    int kk = 0;

    for(MPPlayer p : session.players.values())
    {
        if(p.coordinate != null){

            if(kk != 0){

                List<Vector2f> pLoc = new ArrayList<>();

                int x1 = (int)p.coordinate.x / cellSize;
                int y1 = (int)p.coordinate.y / cellSize;

                int x2 = (int)(p.coordinate.x+p.size) / cellSize
;
                int y2 = (int)p.coordinate.y / cellSize;

                int x3 = (int)p.coordinate.x / cellSize;
                int y3 = (int)(p.coordinate.y+p.size) / cellSize
;

                int x4 = (int)(p.coordinate.x + p.size) / cellSize
;
                int y4 = (int)(p.coordinate.y + p.size) / cellSize
;

                Vector2f xy1 = new Vector2f(x1, y1);
                Vector2f xy2 = new Vector2f(x2, y2);
                Vector2f xy3 = new Vector2f(x3, y3);
                Vector2f xy4 = new Vector2f(x4, y4);


                pLoc.add(xy1);

                if(!xy1.isEqual(xy2)){
                    pLoc.add(xy2);
                    System.out.println("Pridedu 2");
                }

                if(!xy1.isEqual(xy3) && !xy2.isEqual (xy3)){
                    pLoc.add(xy3);
                    System.out.println("Pridedu 3");
                }

                if(!xy1.isEqual(xy4) && !xy2.isEqual(xy4) && !xy3.
isEqual(xy4)){
                    pLoc.add(xy4);
                    System.out.println("Pridedu 4");
                }


                DamgePlayer(p, pLoc, bLoc);

            }

            kk++;
```

```java

                }
            }

    }

    private void DamgePlayer(MPPlayer p, List<Vector2f> pLoc, Vector2f
    bLoc){

            System.out.println("Bombos: " + bLoc.x + " " + bLoc.y);

            for(Vector2f Pxy : pLoc)
            {
                if(bLoc.isEqual(Pxy))
                {
                    System.out.println("Crushed by wall");
                    p.health = 0;
                    return;
                }
            }

    }

}
```

```java
package server;


import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.Timer;
import java.util.TimerTask;

import shared.Vector2f;
import shared.PacketUpdatePlayerPos;

class GameServer
{
  private static GameServer gameServer = null;
//
//          this.stage1builder = new Stage1Builder(gridSize);
//          this.stage2builder = new Stage2Builder(gridSize);
//          this.stage3builder = new Stage3Builder(gridSize);
//


  protected volatile GameBoard gameBoard;

  protected GameState currentState;

  protected volatile Map<Integer, MPPlayer> players;
  protected volatile Network network;
  private GameCycleThread thread;
  private Stage1Factory stage1factory;

private int gameLevel;
private int counter;
private boolean readyForNextFrame;
private Chain chain1;
private Chain chain2;
private Chain chain3;
private Chain chain4;
public boolean enableMovement = false;

public ItemComponent powerUp;

public ItemComponent everyItem;

public Item dash;
public Item teleport;
public Item jump;
public Item slowDown;
public Item past;

public Item bomb;


public Item wall;



public ItemList items;

```

```
private int levelChangeCounter;


private boolean updateBoard = false;


private GameServer()
{

  powerUp = new ItemGroup("Powerup", "Folder that has all powerups
 that have been used");

  everyItem = new ItemGroup("Item list", "This is the list that hold
 data of PowerUps, Walls and Bombs");



  dash = new Item("Dash", "Player moves fast. ", 0);
  teleport = new Item("Teleport", "Player teleports. ", 0);
  slowDown = new Item("SlowDown", "Enemy get slowed down. ", 0);
  jump = new Item("Jump", "Player is capable of jumping over walls. "
 , 0);
  past = new Item("Past", "Player is time traveler. ", 0);


  bomb = new Item("Bomb", "Dangerous item. ", 0);
  wall = new Item("Wall", "Blocks players path. ", 0);

```

```

everyItem.add(powerUp);

powerUp.add(dash);
powerUp.add(teleport);
powerUp.add(slowDown);
powerUp.add(jump);
powerUp.add(past);

everyItem.add(bomb);
everyItem.add(wall);



items = new ItemList(everyItem);



counter = 0;

levelChangeCounter = 0;

//Init Connection
if (!initConnection())
{
   System.err.println("ERROR Connecting to host");
   return;
}

```

```
    gameLevel = 2;

    this.players = new HashMap<Integer, MPPlayer>();

    setGameLevel();

    this.thread = new GameCycleThread();
    this.thread.start();

    chain1 = new GenerateWalls();
    chain3 = new GenerateDesWall();
    chain2 = new GenerateTrap();
    readyForNextFrame = true;
    chain1.setNextChain(chain2);
    chain2.setNextChain(chain3);

    GameState waitForPlayersState = new StateWaitForPlayers();
    GameState playGameState = new StatePlayGame();
    GameState endingGameState = new StateEndingGame();
    GameState changingMapState = new StateChangingMap();

    waitForPlayersState.setNextState(playGameState);
    playGameState.setNextState(endingGameState);
    endingGameState.setNextState(changingMapState);
    changingMapState.setNextState(waitForPlayersState);
    currentState = waitForPlayersState;


  }
```

```java

protected GameServer getContext()
{
  return this;
}

public boolean checkIfPlayerDead(){
  for(MPPlayer p : players.values())
  {
    if(p.coordinate != null){
      if(p.health < 1 && p.canDie){
        p.deathCounter +=1;
        System.out.println("Mires");
        //setGameLevel();
        return true;
      }
    }
  }
  return false;

}

public String getWinner(){
  boolean firstCycle = true;
  for(MPPlayer p : players.values())
  {
    if (!firstCycle)
    {
      if(p.coordinate != null){
```

```
                if(p.health > 0){
                    //setGameLevel();
                    return String.valueOf(p.c.getID());
                }
            }
        }
        firstCycle = false;
    }
    return "None";

}

public int getConnectedPlayerCount(){
    return players.size();

}
//nustatome sekancia busena
public void setState(GameState nextState)
{
    this.currentState = nextState;
}


public void setGameLevel(){



    if(gameLevel == 3)
        gameLevel = 1;
    else
      gameLevel++;


    for(MPPlayer p : players.values())
    {
      if(p.coordinate != null){
        System.out.println("Atstatau gyvybes");
        p.health = p.baseHealth;
        p.speed = p.baseSpeed;

      }
    }



    ChangeLevel(gameLevel);

  }
  public void stopPlayers()
  {
    for(MPPlayer p : players.values())
    {
      if(p.coordinate != null){
        p.isHoldingDown = false;
        p.isHoldingUp = false;
        p.isHoldingLeft = false;
        p.isHoldingRight = false;
        p.isHoldingSkill = false;
        p.isHoldingUse = false;
      }


    }
  }

  public void ChangeLevel(int level){

    levelChangeCounter = 60 * 2;

    stopPlayers();
    gameLevel = level;

    IStageBuilder builder;
    AbstractFactory factory;
    switch (level){
      case 1:
      default:
        builder = new Stage1Builder(20);
        factory = new Stage1Factory();
        this.gameBoard = new GameBoard(factory, builder, 1);
        break;
      case 2:
        builder = new Stage2Builder(20);
        factory = new Stage1Factory();
        this.gameBoard = new GameBoard(factory, builder, 2);
        break;
      case 3:
```

```
                builder = new Stage3Builder(20);
                factory = new Stage1Factory();
                this.gameBoard = new GameBoard(factory, builder, 3);
                break;
        }
        respawnAllPlayers();
        refreshMapForAllPlayers();



    }

    public static GameServer getInstance()
    {
        if (gameServer == null)
        {
            gameServer = new GameServer();
        }
        return gameServer;
    }

    private boolean initConnection()
    {
        this.network = new Network();

        if (!this.network.initKryoServer())
        {
            return false;
        }

        return true;
    }

    public void respawnAllPlayers()
    {
        for(MPPlayer p : players.values())
        {
            if(p.coordinate != null){
                this.respawnPlayer(p);
            }


        }

    }

    public void respawnPlayer(PlayerInfo p)
    {
        GameServer gameserver = GameServer.getInstance();
        boolean teleported = false;
        int maxRetry = 60;
        int retry = 0;
        while (!teleported && retry < maxRetry)
        {
            int randomCoordX = getRandomNumber (0, gameserver.gameBoard.
      gridSize);
            int randomCoordY = getRandomNumber (0, gameserver.gameBoard.
      gridSize);
```

```
            if (gameserver.gameBoard.objects[randomCoordX][randomCoordY]
    instanceof Ground)
            {
                p.coordinate.x = randomCoordX * (gameserver.gameBoard.size /
    gameserver.gameBoard.gridSize);
                p.coordinate.y = randomCoordY * (gameserver.gameBoard.size /
    gameserver.gameBoard.gridSize);
                teleported = true;
            }
            retry++;
        }
    }

    private int getRandomNumber(int min, int max)
    {
        return (int) ((Math.random() * (max - min)) + min);
    }

    public void updatePlayer(int id, PacketUpdatePlayerPos playerPacket)
    {
        MPPlayer player = players.get(id);
        if (player != null && levelChangeCounter < 0)
        {
            player.isHoldingUp = playerPacket.isHoldingUp != null ?
    playerPacket.isHoldingUp : player.isHoldingUp;
            player.isHoldingDown = playerPacket.isHoldingDown != null ?
    playerPacket.isHoldingDown : player.isHoldingDown;
            player.isHoldingLeft = playerPacket.isHoldingLeft != null ?
    playerPacket.isHoldingLeft : player.isHoldingLeft;
```

```
      player.isHoldingRight = playerPacket.isHoldingRight != null ?
    playerPacket.isHoldingRight : player.isHoldingRight;
      player.isHoldingUse = playerPacket.isHoldingUse != null ?
    playerPacket.isHoldingUse : player.isHoldingUse;
      player.isHoldingSkill = playerPacket.isHoldingSkill != null ?
    playerPacket.isHoldingSkill : player.isHoldingSkill;
      players.put(player.id, player);
    }
  }

  public void addPlayer(MPPlayer player)
  {
    this.players.put(player.c.getID(), player);
    this.respawnPlayer(player);
    this.network.sendGameBoard(gameBoard, player);

  }

  public void refreshMapForAllPlayers()
  {
    for(MPPlayer p : players.values())
    {
      network.sendPlayerInfo(p, true);
      this.network.sendGameBoard(gameBoard, p);
    }
  }

  public void removePlayer(int id)
  {

    players.remove(id);
  }


    private class GameCycleThread extends Thread
    {
      volatile boolean isGameRunning = true;
      private final int gameSpeed = 16; //The lower the number the
   faster the game is



      public GameCycleThread()
      {
        this.isGameRunning = true;
      }

        public void run()
        {
            long lastTime = System.nanoTime();
            final double ns = 1000000000.0 / 60.0;
            double delta = 0;
            while(true){
                long now = System.nanoTime();
                delta += (now - lastTime) / ns;
                lastTime = now;
                while(delta >= 1){
            update();
```

```
                    delta--;
                }
            }
    }

    public void stopGame()
    {
        this.isGameRunning = false;

        //network should probably be closed by the parent
        network.close();
    }

    private void update()
    {

  counter++;
  levelChangeCounter--;
  currentState.handleUpdate();
  if (currentState.isReadyForNextStage())
  {
    currentState.getNextState(getContext());
  }

  if (enableMovement)
  {
        updatePlayers();
        gameBoard.runTick();
  }
```

```
        surenkamTeksta();


        checkIfPlayerDead();




            }



    private void surenkamTeksta(){
        InputStreamReader fileInputStream=new InputStreamReader(System.in
    );
        BufferedReader bufferedReader=new BufferedReader(fileInputStream)
    ;

        String tekstas = "nieko nenuskaityta";


        try {
            if (bufferedReader.ready()){

                tekstas = bufferedReader.readLine();



                System.out.println("irasytas tekstas " + tekstas);
```

```java



        String tekstasLower = tekstas.toLowerCase();


        String[] sarasas = tekstasLower.split(" ");


        if(sarasas.length == 1){

            items.getItemList();

        }
        else if(sarasas.length > 3){
            ConversionContext task = new ConversionContext(tekstas);

            String whatToPlace = task.getWhat();
            String whereToPlace = task.getWhere();
            int howManyToPlace = task.getQuantity();


            Task newTask = new Task(howManyToPlace, whatToPlace,
whereToPlace, gameBoard);

            gameBoard = chain1.result(newTask);

            System.out.println(whatToPlace + " How many: " +
howManyToPlace);
```

```
        } else if(sarasas.length > 1){

          if(sarasas[0].equals("change") && sarasas[1].equals("level"
))){
            int lygis;
            try {
              lygis = Integer.parseInt(sarasas[2]);

              if(lygis > 0 && lygis < 4){
                ChangeLevel(lygis);

              }else {
                System.out.println("Toks lygis neegzistuoja: " +
lygis);
              }

            } catch (NumberFormatException e) {
              System.out.println("Toks lygis neegzistuoja: " +
sarasas[2]);
            }
          }
          else {
            System.out.println("Unknown command " + tekstas);
          }


        } else
        {
          System.out.println("Unknown command " + tekstas);
```

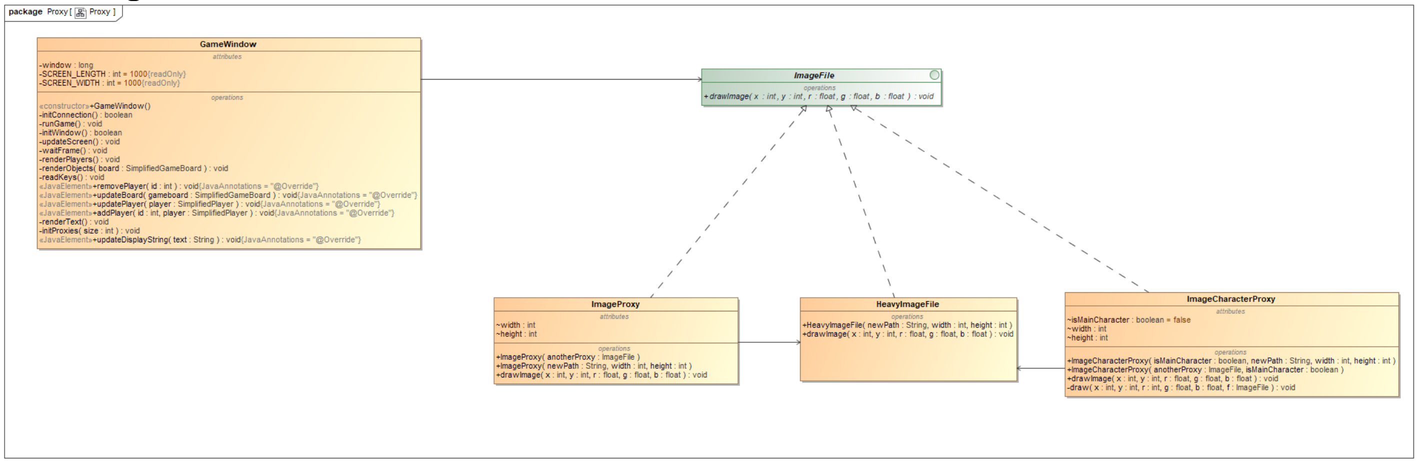
```
                }




            }


        } catch (IOException e) {
            e.printStackTrace();
        }










    }


        private void updatePlayers()
        {
            int k = 0;
        for(MPPlayer p : players.values())
        {
            if (k!= 0)
            {
                if (p.isHoldingSkill)
                {
                    p.tryUsingSpell();
                }

                p.onTick();
            p.reduceTimer();

            if (p.isHoldingUse)
            {

                if(p.bombTimer < 0){
                    gameBoard.SpawnBomb(p);

                    p.setBombTimer(2);
                }

            }
            p.coordinate = checkCollision(p);
            network.sendGameBoard(gameBoard, p);
                network.sendPlayerInfo(p, true);
            }
            k++;
        }
        }

        private Vector2f checkCollision(MPPlayer p)
        {

            Vector2f coordsAfterMove = new Vector2f(p.coordinate.x, p.
    coordinate.y);

            float padding = 0.001f;
        float cellSize = gameBoard.cellSize();

        boolean moveX = true;
        boolean moveY = true;

        if (p.isHoldingLeft)
        {
          coordsAfterMove.x -= p.speed;
        }

        if (p.isHoldingRight)
        {

          coordsAfterMove.x += p.speed;
        }

        boolean collidingLeft = ((int)coordsAfterMove.x / cellSize -
    padding)<((int)p.coordinate.x / cellSize);
        boolean collidingRight=(((int)coordsAfterMove.x+p.size+
    padding)/cellSize)>(((int)p.coordinate.x+p.size)/cellSize);
        boolean isCollidingX = collidingLeft || collidingRight;
        moveX = !(coordsAfterMove.x <= 0 || coordsAfterMove.x >=
    gameBoard.size - p.size);
        //Some smoothing when going around edges would be nice
        if (isCollidingX && moveX)
        {
          int x = 0, y = 0, y1 = 0;
          if (collidingLeft)
          {
            x = (int) ((coordsAfterMove.x) / cellSize);
            y = (int) (p.coordinate.y / cellSize);
            y1 = (int) ((p.coordinate.y + p.size) / cellSize);

          }
          if (collidingRight)
          {
            x = (int) ((coordsAfterMove.x + p.size) / cellSize);
            y = (int) (p.coordinate.y / cellSize);
            y1 = (int) ((p.coordinate.y + p.size) / cellSize);

          }

          if (y == y1)
          {
            moveX = gameBoard.objects[x][y].isWalkable;
            gameBoard.objects[x][y].onStep(p);
          }
          else
          {
            moveX= gameBoard.objects[x][y].isWalkable && gameBoard.
    objects[x][y1].isWalkable;
            gameBoard.objects[x][y].onStep(p);
            gameBoard.objects[x][y1].onStep(p);
          }
        }


        if (p.isHoldingUp)
        {
          coordsAfterMove.y += p.speed;
        }

        if (p.isHoldingDown)
        {
          coordsAfterMove.y -= p.speed;
        }

        boolean collidingUp = (((int)coordsAfterMove.y + p.size - padding
    ) / cellSize) > (((int)p.coordinate.y + p.size) / cellSize);
        boolean collidingDown = ((int)coordsAfterMove.y/cellSize+
    padding)< ((int)p.coordinate.y / cellSize);
        boolean isCollidingY = collidingUp || collidingDown;
        moveY = !(coordsAfterMove.y <= 0 || coordsAfterMove.y >=
    gameBoard.size - p.size);

        //Some smoothing when going around edges would be nice
        if (isCollidingY && moveY)
        {
          int x = 0, x1 = 0, y = 0;

          if (collidingUp)
          {
            y = (int) ((coordsAfterMove.y + p.size) / cellSize);
            x = (int) (p.coordinate.x / cellSize);
            x1 = (int) ((p.coordinate.x + p.size) / cellSize);

        }
        if (collidingDown)
        {
            y = (int) ((coordsAfterMove.y) / cellSize);
            x = (int) (p.coordinate.x / cellSize);
            x1 = (int) ((p.coordinate.x + p.size) / cellSize);
        }

        if (x == x1)
        {
            moveY = gameBoard.objects[x][y].isWalkable;
            gameBoard.objects[x][y].onStep(p);
        }
        else
        {
            moveY = gameBoard.objects[x][y].isWalkable && gameBoard.
    objects[x1][y].isWalkable;
            gameBoard.objects[x][y].onStep(p);
            gameBoard.objects[x1][y].onStep(p);
        }
    }

    coordsAfterMove.x = moveX ? coordsAfterMove.x : p.coordinate.x;
    coordsAfterMove.y = moveY ? coordsAfterMove.y : p.coordinate.y;

    return coordsAfterMove;
```

```

                }

        }

}
```

Proxy šablonas

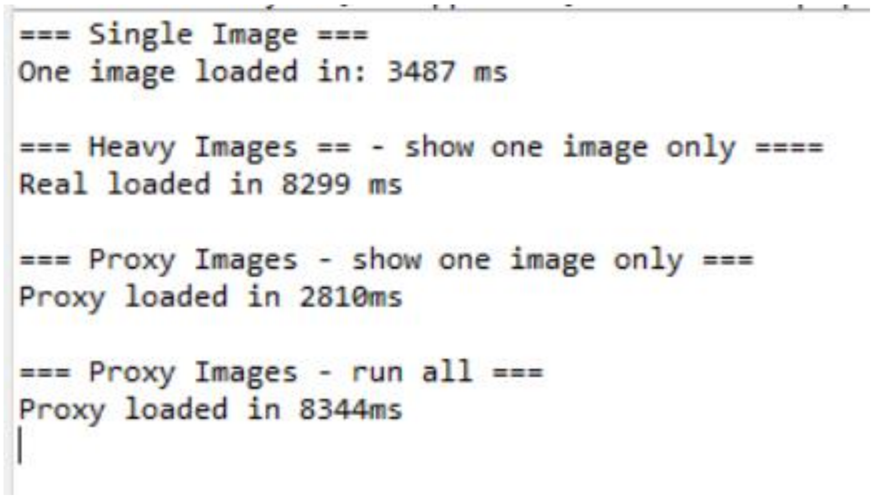
Kuriant projektą prireikė efektyvaus būdo apdoroti nuotraukas, kad nereikėtų į atmintį iš naujo įkėlinėti ir naudoti tik tada kada reikia. Taip pat kilo noras tekstūras šiek tiek modifikuoti jau įkėlus į atmintį. Šiuos pageidavimus tikslingai padeda įvykdyti Proxy šablonas

UML diagrama:



package Proxy [Proxy]
GameWindow
attributes
-window : long
-SCREEN_LENGTH : int = 1000{readOnly}
-SCREEN_WIDTH : int = 1000{readOnly}
operations
«constructor»+GameWindow()
-initConnection() : boolean
-runGame() : void
-initWindow() : boolean
-updateScreen() : void
-waitFrame() : void
-renderPlayers() : void
-renderObjects(board : SimplifiedGameBoard) : void
-readKeys() : void
«JavaElement»+removePlayer(id : int) : void{JavaAnnotations = "@Override"}
«JavaElement»+updateBoard(gameboard : SimplifiedGameBoard) : void{JavaAnnotations = "@Override"}
«JavaElement»+updatePlayer(player : SimplifiedPlayer) : void{JavaAnnotations = "@Override"}
«JavaElement»+addPlayer(id : int, player : SimplifiedPlayer) : void{JavaAnnotations = "@Override"}
-renderText() : void
-initProxies(size : int) : void
«JavaElement»+updateDisplayString(text : String) : void{JavaAnnotations = "@Override"}
ImageFile
operations
+drawImage(x : int, y : int, r : float, g : float, b : float) : void
ImageProxy
attributes
~width : int
~height : int
operations
+ImageProxy(anotherProxy : ImageFile)
+ImageProxy(newPath : String, width : int, height : int)
+drawImage(x : int, y : int, r : float, g : float, b : float) : void
HeavyImageFile
operations
+HeavyImageFile(newPath : String, width : int, height : int)
+drawImage(x : int, y : int, r : float, g : float, b : float) : void
ImageCharacterProxy
attributes
~isMainCharacter : boolean = false
~width : int
~height : int
operations
+ImageCharacterProxy(isMainCharacter : boolean, newPath : String, width : int, height : int)
+ImageCharacterProxy(anotherProxy : ImageFile, isMainCharacter : boolean)
+drawImage(x : int, y : int, r : float, g : float, b : float) : void
-draw(x : int, y : int, r : int, g : float, b : float, f : ImageFile) : void

Greitaveika lyginant naudojima be šio šablono:


```
=== Single Image ===
One image loaded in: 3487 ms

=== Heavy Images == - show one image only ====
Real loaded in 8299 ms

=== Proxy Images - show one image only ===
Proxy loaded in 2810ms

=== Proxy Images - run all ===
Proxy loaded in 8344ms
```

Kaip matome skirtumas labia didelis

Kodas

```
package client;

public class ImageCharacterProxy implements ImageFile{

  ImageFile file;

  String path;

  boolean isMainCharacter = false;
  String user = null;
  int width;
  int height;
  ImageFile parentProxy;

  public ImageCharacterProxy(boolean isMainCharacter, String newPath,
   int width, int height)
  {
    this.isMainCharacter = isMainCharacter;
    this.path = newPath;
    this.width = width;
    this.height = height;
```

```
}

public ImageCharacterProxy(ImageFile anotherProxy, boolean
 isMainCharacter){
  parentProxy = anotherProxy;
  this.isMainCharacter = isMainCharacter;
}


@Override
public void drawImage(int x, int y, float r, float g, float b)
{
  if(parentProxy == null)
  {
    if(file == null)
    {
      file = new HeavyImageFile(this.path, this.width, this.height);
      this.draw(x, y, y, g, b, file);
    }
    else
    {
      this.draw(x, y, y, g, b, file);
    }

  }
  else
  {
    this.draw(x, y, y, g, b, parentProxy);
  }
```

```java
    }

    private void draw(int x, int y, int r, float g, float b, ImageFile f)
    {

        if(isMainCharacter)
        {
            //Removing color for main char
            f.drawImage(x, y, 1f, 1f, 1f);
        }
        else
        {
            f.drawImage(x, y, r, g, b);
            float size = 40;
            float xEn = x*4/size;
            float yEn = y*4/size;
            Text.drawString("enemy",xEn- 8/10 ,yEn-1 , size, 2);
        }
    }
}
```

```java
package client;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;

```

```java

public class HeavyImageFile implements ImageFile {

  Sprite heavySprite;
  String path;
  private TextureLoader textureLoader;

  public HeavyImageFile(String newPath, int width, int height)
  {
    textureLoader = new TextureLoader();
    this.heavySprite = new Sprite(textureLoader, newPath, width, height
    );
  }

  @Override
  public void drawImage(int x, int y, float r, float g, float b) {
     this.heavySprite.draw(x, y, r, g, b);

  }

}
```

```java
package client;

public class ImageProxy implements ImageFile{

  HeavyImageFile file = null; //heavy image file
  int width;
  int height;
```

```
//HeavyImageFile file;

String path;
ImageFile parentProxy;

public ImageProxy(String newPath, int width, int height)
{
    this.path = newPath;
    this.width = width;
    this.height = height;
}

public ImageProxy(ImageFile anotherProxy){
    parentProxy = anotherProxy;
}

@Override
public void drawImage(int x, int y, float r, float g, float b)
{
    if(parentProxy == null)
    {
        if(file == null)
        {
            file = new HeavyImageFile(path, width, height);
        }
        file.drawImage(x, y, r, g, b);
    }
    else
    {
```

```java
            parentProxy.drawImage(x, y, r, g, b);
        }

    }


}
```

```java
package client;


public interface ImageFile
{

    public void drawImage(int x, int y, float r, float g, float b);

}
```

```java
package client;

import java.nio.ByteBuffer;
import java.util.HashMap;
import java.util.Map;
import org.lwjgl.glfw.GLFW;
import org.lwjgl.opengl.GL11;

import shared.SimplifiedGameBoard;
import shared.SimplifiedPlayer;
import shared.Vector2f;
import shared.ObjectType;

import org.lwjgl.opengl.GL;



public class GameWindow implements UpdateGameDataDelegate
{
    private long window;
    final private int SCREEN_LENGTH = 1000;
    final private int SCREEN_WIDTH = 1000;
    private SimplifiedGameBoard board;
    private Network network;
    private SimplifiedPlayer mainPlayer;
    private Map<Integer, ClientPlayer> players;
    private String displayString = null;
    //Textures
    private ImageFile           bombSprite;
    private ImageFile           wallSprite;
    private ImageFile           groundSprite;
    private ImageFile           powerupSprite;
    private ImageFile           trapSprite;
    private ImageFile           fireSprite;
    private ImageFile           playerSprite;

    public GameWindow()
    {

        //Init Window
        if (!initWindow())
        {
          System.err.println("ERROR WHILE INITIATING WINDOW");
          return;
        }

        //Init Connection
        if (!initConnection())
        {
          System.err.println("ERROR Connecting to host");
          return;
        }


        this.mainPlayer = new SimplifiedPlayer(this.network.client.getID())
       ;
        this.players=new HashMap<Integer,Client Player >();
        runGame();
      }


    private void initProxies(int size)
    {
        bombSprite = new ImageProxy("Tnt.png", size, size);
        wallSprite = new ImageProxy("Cobble.png",size,size);
        groundSprite = new ImageProxy("Grass.png",size,size);
        powerupSprite = new ImageProxy("Powerup.png",size,size);
        trapSprite = new ImageProxy("Trap.png", size,size);
        fireSprite = new ImageProxy("Fire.png",size,size);
        playerSprite = new ImageProxy("Player.jpg", 40, 40);
```

```java
}

private boolean initConnection()
{
    this.network = new Network();

    if (!this.network.connect(this))
    {
        return false;
    }

    return true;
}

private void runGame()
{
    while (GLFW.glfwWindowShouldClose(window) != true)
    {
        readKeys();
        updateScreen();
        waitFrame();
    }
}

private boolean initWindow()
{
    if (GLFW.glfwInit() != true)
    {
        return false;
    }

    window = GLFW.glfwCreateWindow(SCREEN_LENGTH, SCREEN_WIDTH, "Window
  ",0, 0);

    GLFW.glfwShowWindow(window);
    GLFW.glfwMakeContextCurrent(window);
    GL.createCapabilities();

    //Init GL
    // enable textures since we're going to use these for our sprites
    GL11.glEnable(GL11.GL_TEXTURE_2D);
    GL11.glEnable(GL11.GL_BLEND);
    GL11.glDepthFunc(GL11.GL_ALWAYS);
    GL11.glBlendFunc(GL11.GL_SRC_ALPHA, GL11.GL_ONE_MINUS_SRC_ALPHA);
    // disable the OpenGL depth test since we're rendering 2D graphics
    GL11.glEnable(GL11.GL_DEPTH_TEST);
    GL11.glMatrixMode(GL11.GL_PROJECTION);
    GL11.glLoadIdentity();
    GL11.glOrtho(0, SCREEN_LENGTH, 0, SCREEN_WIDTH, -1, 1);
    GL11.glMatrixMode(GL11.GL_MODELVIEW);
    return true;
}
private void updateScreen()
{
    GLFW.glfwPollEvents();
    GL11.glClear(GL11.GL_COLOR_BUFFER_BIT | GL11.GL_DEPTH_BUFFER_BIT);
    GL11.glEnable(GL11.GL_TEXTURE_2D);
    renderObjects(board);
    renderPlayers();
    GL11.glDisable(GL11.GL_TEXTURE_2D);
    renderText();
    GLFW.glfwSwapBuffers(window);
}

private void renderText()
{
    Text.drawString("Spell " + this.mainPlayer.skillName + " Cooldown
    " + (int)(this.mainPlayer.skillCooldown / 60), 5, 2, 40, 2);
    Text.drawString("Player" + this.mainPlayer.id + " Health " + this.
    mainPlayer.health, 45, 2, 40, 2);

    int size = 40;



    if(this.mainPlayer.health < 1){

        Text.drawString("GG", 2, 6, 100, 20);
    }

    for(SimplifiedPlayer mpPlayer : players.values())
    {

        if(this.mainPlayer.id != mpPlayer.id){

            if (mpPlayer.coordinate != null)
            {
```

```
          float x = mpPlayer.coordinate.x*4/size;
          float y = mpPlayer.coordinate.y*4/size;

        }

      }

    }

    if (this.displayString != null)
    {
      Text.drawString(this.displayString, 10, 40, 55, 4);
    }


  }

  //bad design
  private void waitFrame()
  {
    try
    {
      //around 120 tiems a second
      Thread.sleep(1L);
    }
    catch (InterruptedException e)
    {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
}
private void renderPlayers()
{
    for(ClientPlayer mpPlayer : players.values()){
        mpPlayer.getSprite().drawImage((int)mpPlayer.coordinate.x, (int)
  mpPlayer.coordinate.y,0f,(float)(255/255),(float)(127/255));
    }
}


private void renderObjects(SimplifiedGameBoard board)
{
    GL11.glEnable(GL11.GL_BLEND);
    GL11.glBlendFunc(GL11.GL_SRC_ALPHA, GL11.GL_ONE_MINUS_SRC_ALPHA);
    if (board == null)
    {
        return;
    }
    int sizeX = SCREEN_LENGTH / board.gridSize;
    int sizeY = SCREEN_WIDTH / board.gridSize;


    for (int i = 0; i < board.gridSize; i++) {
        for (int j = 0; j < board.gridSize; j++) {
            float red = (float)Integer.valueOf(board.objects[i][j].color.
  substring(1,3), 16)/255;
            float green = (float)Integer.valueOf(board.objects[i][j].color.
  substring(3,5),16)/255;
```

```
        float blue = (float)Integer.valueOf(board.objects[i][j].color.
  substring(5,7), 16)/255;
        ImageProxy drawer;
        switch(board.objects[i][j].type)
        {
          default:
          case GROUND:
            drawer = new ImageProxy(groundSprite);
          break;
          case WALL:
            drawer = new ImageProxy(wallSprite);
          break;
          case TRAP:
            drawer = new ImageProxy(trapSprite);
          break;
          case POWERUP:
            drawer = new ImageProxy(powerupSprite);
          break;
          case BOMB:
            drawer = new ImageProxy(bombSprite);
          break;
        }

        drawer.drawImage(i*sizeX, j*sizeY, red, green, blue);
        GL11.glDisable(GL11.GL_BLEND);
        if (board.objects[i][j].explodeAnimation == true)
        {
          GL11.glBlendFunc(GL11.GL_SRC_ALPHA, GL11.
GL_ONE_MINUS_SRC_ALPHA);
```

```
                drawer = new ImageProxy(fireSprite);
                drawer.drawImage(i*sizeX, j*sizeY, 1, 1, 1);
            }
        }
    }



}

private void readKeys()
{

    //TODO should read from events instead of polling
    //https://www.glfw.org/docs/3.3/input_guide.html




    if (GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_W) == GLFW.GLFW_TRUE)
    {
        if (!this.mainPlayer.isHoldingUp)
        {
            this.network.sendPacketButtonPressUp();
        }
    }
    else
    {
        if (this.mainPlayer.isHoldingUp)
        {
            this.network.sendPacketButtonReleaseUp();
        }

    }

    if (GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_S) == GLFW.GLFW_TRUE)
    {
        if (!this.mainPlayer.isHoldingDown)
        {
            this.network.sendPacketButtonPressDown();
        }
    }
    else
    {
        if (this.mainPlayer.isHoldingDown)
        {
            this.network.sendPacketButtonReleaseDown();
        }

    }

    if (GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_A) == GLFW.GLFW_TRUE)
    {
        if (!this.mainPlayer.isHoldingLeft)
        {
            this.network.sendPacketButtonPressLeft();
        }
    }
```

```
        else
        {
            if (this.mainPlayer.isHoldingLeft)
            {
                this.network.sendPacketButtonReleaseLeft();
            }

        }

        if(GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_D) == GLFW.GLFW_TRUE)
        {
            if (!this.mainPlayer.isHoldingRight)
            {
                this.network.sendPacketButtonPressRight();
            }
        }
        else
        {
            if (this.mainPlayer.isHoldingRight)
            {
                this.network.sendPacketButtonReleaseRight();
            }

        }


        if(GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_E) == GLFW.GLFW_TRUE)
        {
            if (!this.mainPlayer.isHoldingUse)
            {
                this.network.sendPacketButtonPressUse();
            }
        }
        else
        {
            if (this.mainPlayer.isHoldingUse)
            {
                this.network.sendPacketButtonReleaseUse();
            }
        }

        if(GLFW.glfwGetKey(this.window, GLFW.GLFW_KEY_SPACE) == GLFW.
    GLFW_TRUE)
        {
            if (!this.mainPlayer.isHoldingSkill)
        {
                this.network.sendPacketButtonPressSkill();
            }
        }
        else
        {
            if (this.mainPlayer.isHoldingSkill)
            {
                this.network.sendPacketButtonReleaseSkill();
            }
        }

        if (GLFW.glfwGetKey(window, GLFW.GLFW_KEY_ESCAPE) == GLFW.GLFW_TRUE
     )
      {
        GLFW.glfwSetWindowShouldClose(window, true);
      }
    }

    @Override
    public void updatePlayer(SimplifiedPlayer player)
    {
      if (this.mainPlayer.id == player.id)
      {
        ClientPlayer clientPlayer = new ClientPlayer(player, playerSprite
     , true);
        this.mainPlayer = player;
        this.players.put(player.id, clientPlayer);
      }
      else
      {
        ClientPlayer clientPlayer = new ClientPlayer(player, playerSprite
     , false);
        this.players.put(player.id, clientPlayer);
      }

    }
    @Override
    public void addPlayer(int id, SimplifiedPlayer player)
    {
      if (player.id == mainPlayer.id)
      {
```

```
            ClientPlayer clientPlayer = new ClientPlayer(player, playerSprite
        , true);
            players.put(id, clientPlayer);
        }
        else
        {
            ClientPlayer clientPlayer = new ClientPlayer(player, playerSprite
        , false);
            players.put(id, clientPlayer);
        }
    }

    @Override
    public void removePlayer(int id)
    {
        players.remove(id);
    }

    @Override
    public void updateBoard(SimplifiedGameBoard gameboard)
    {
        if ( this.board == null)
        {
            this.initProxies(gameboard.size /gameboard.gridSize);
        }
        this.board = gameboard;

    }

```

```

  @Override
  public void updateDisplayString(String text)
  {
    this.displayString = text;

  }

}
```

```
package client;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.HashMap;

import javax.imageio.ImageIO;

import org.lwjgl.glfw.GLFW;
import org.lwjgl.opengl.GL;
import org.lwjgl.opengl.GL11;

import shared.SimplifiedPlayer;


public class ProxyTest {

  public static void main(String[] args) {
    if (GLFW.glfwInit() != true)
    {
        return;
    }

    long window;
    int SCREEN_LENGTH = 1000;
    int SCREEN_WIDTH = 1000;
    //Init Window
    window = GLFW.glfwCreateWindow(SCREEN_LENGTH, SCREEN_WIDTH, "Window
",0, 0);

   GLFW.glfwShowWindow(window);
   GLFW.glfwMakeContextCurrent(window);
   GL.createCapabilities();

    //Init GL
    // enable textures since we're going to use these for our sprites
    GL11.glEnable(GL11.GL_TEXTURE_2D);
    GL11.glEnable(GL11.GL_BLEND);
    GL11.glDepthFunc(GL11.GL_ALWAYS);
    GL11.glBlendFunc(GL11.GL_SRC_ALPHA, GL11.GL_ONE_MINUS_SRC_ALPHA);
    // disable the OpenGL depth test since we're rendering 2D graphics
    GL11.glEnable(GL11.GL_DEPTH_TEST);
    GL11.glMatrixMode(GL11.GL_PROJECTION);
    GL11.glLoadIdentity();
    GL11.glOrtho(0, SCREEN_LENGTH, 0, SCREEN_WIDTH, -1, 1);
    GL11.glMatrixMode(GL11.GL_MODELVIEW);

```

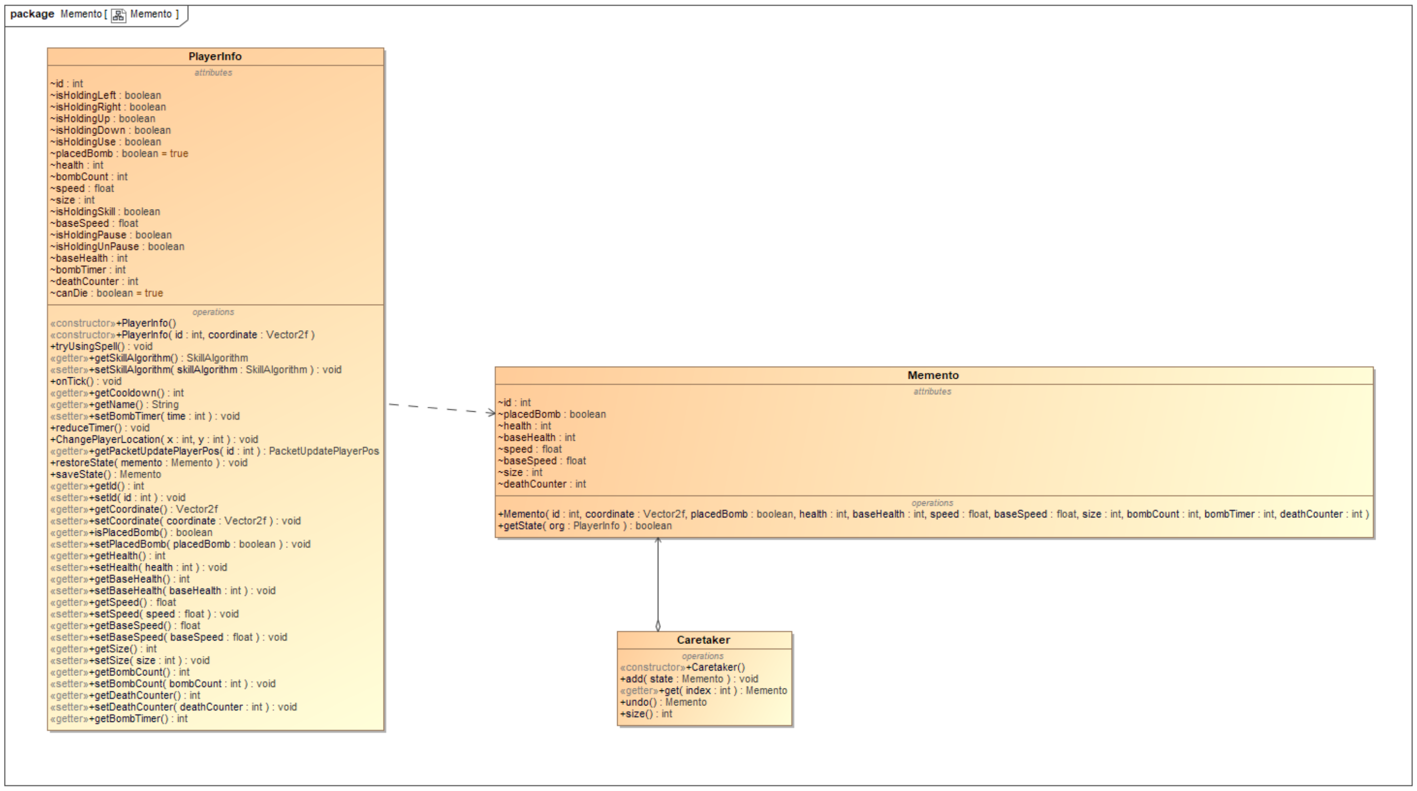
```


  String path = "Test.jpg";

  System.out.println("=== Single Image ===");
  long startP = System.currentTimeMillis();
  TextureLoader loader = new TextureLoader();
  Sprite newsprite = new Sprite(loader,path,50,50);

  long stopP = System.currentTimeMillis();

  System.out.println("One image loaded in: " + (stopP -startP)+"
 ms");

  System.gc(); //   Runtime.getRuntime().gc();

  System.out.println("\n=== Heavy Images == - show one image only
====");
  long startReal = System.currentTimeMillis();
  ImageFile h1 = new HeavyImageFile(path,50,50);
  ImageFile h2 = new HeavyImageFile(path,50,50);
  ImageFile h3 = new HeavyImageFile(path,50,50);
  h2.drawImage(0, 0, 1,1,1);
  long stopReal = System.currentTimeMillis();
  System.out.println("Real loaded in " + (stopReal -startReal)+"
 ms");

  System.gc(); //   Runtime.getRuntime().gc();

```

```
    System.out.println("\n=== Proxy Images - show one image only ===");
    long startProxy = System.currentTimeMillis();
    ImageFile p1 = new ImageProxy(path,50,50);
    ImageFile p2 = new ImageProxy(path,50,50);
    ImageFile p3 = new ImageProxy(path,50,50);
    p2.drawImage(0, 0, 1,1,1);
    long stopProxy = System.currentTimeMillis();
    System.out.println("Proxy loaded in " + (stopProxy - startProxy) +
  "ms");

    System.gc(); //    Runtime.getRuntime().gc();

    System.out.println("\n=== Proxy Images - run all ===");
    long startProxyA = System.currentTimeMillis();
    ImageFile a1 = new ImageProxy(path,50,50);
    ImageFile a2 = new ImageProxy(path,50,50);
    ImageFile a3 = new ImageProxy(path,50,50);
    a1.drawImage(0, 0, 1,1,1);;
    a2.drawImage(0, 0, 1,1,1);;
    a3.drawImage(0, 0, 1,1,1);;
    long stopProxyA = System.currentTimeMillis();
    System.out.println("Proxy loaded in " + (stopProxyA - startProxyA)
 + "ms");
  }

}
```

Memento

Kilo idėja į žaidimą įdėti įgūdį, kuris leidžia vartotojui “grįžti laiku” ir nekreipti dėmesio į klaidas, kurias padarė. Šiai idėjai tikslingai pravertė memento šablonas.

UML diagrama:



package Memento [Memento]
PlayerInfo
attributes
~id : int
~isHoldingLeft : boolean
~isHoldingRight : boolean
~isHoldingUp : boolean
~isHoldingDown : boolean
~isHoldingUse : boolean
~placedBomb : boolean = true
~health : int
~bombCount : int
~speed : float
~size : int
~isHoldingSkill : boolean
~baseSpeed : float
~isHoldingPause : boolean
~isHoldingUnPause : boolean
~baseHealth : int
~bombTimer : int
~deathCounter : int
~canDie : boolean = true
operations
«constructor»+PlayerInfo()
«constructor»+PlayerInfo(id : int, coordinate : Vector2f)
+tryUsingSpell() : void
«getter»+getSkillAlgorithm() : SkillAlgorithm
«setter»+setSkillAlgorithm(skillAlgorithm : SkillAlgorithm) : void
+onTick() : void
«getter»+getCooldown() : int
«getter»+getName() : String
«setter»+setBombTimer(time : int) : void
+reduceTimer() : void
+ChangePlayerLocation(x : int, y : int) : void
«getter»+getPacketUpdatePlayerPos(id : int) : PacketUpdatePlayerPos
+restoreState(memento : Memento) : void
+saveState() : Memento
«getter»+getId() : int
«setter»+setId(id : int) : void
«getter»+getCoordinate() : Vector2f
«setter»+setCoordinate(coordinate : Vector2f) : void
«getter»+isPlacedBomb() : boolean
«setter»+setPlacedBomb(placedBomb : boolean) : void
«getter»+getHealth() : int
«setter»+setHealth(health : int) : void
«getter»+getBaseHealth() : int
«setter»+setBaseHealth(baseHealth : int) : void
«getter»+getSpeed() : float
«setter»+setSpeed(speed : float) : void
«getter»+getBaseSpeed() : float
«setter»+setBaseSpeed(baseSpeed : float) : void
«getter»+getSize() : int
«setter»+setSize(size : int) : void
«getter»+getBombCount() : int
«setter»+setBombCount(bombCount : int) : void
«getter»+getDeathCounter() : int
«setter»+setDeathCounter(deathCounter : int) : void
«getter»+getBombTimer() : int
Memento
attributes
~id : int
~placedBomb : boolean
~health : int
~baseHealth : int
~speed : float
~baseSpeed : float
~size : int
~deathCounter : int
operations
+Memento(id : int, coordinate : Vector2f, placedBomb : boolean, health : int, baseHealth : int, speed : float, baseSpeed : float, size : int, bombCount : int, bombTimer : int, deathCounter : int)
+getState(org : PlayerInfo) : boolean
Caretaker
operations
«constructor»+Caretaker()
+add(state : Memento) : void
«getter»+get(index : int) : Memento
+undo() : Memento
+size() : int

Kodas:

```

import shared.Vector2f;

public class Memento {

  int id;
  Vector2f coordinate;
  boolean placedBomb;
  int health;
  int baseHealth;
  float speed;
  float baseSpeed;
  int size;
  int deathCounter;



  public Memento( int id,
    Vector2f coordinate,
    boolean placedBomb, int health,
    int baseHealth, float speed, float baseSpeed,
    int size, int bombCount, int bombTimer, int deathCounter)
  {
    this.id = id;
    this.coordinate = coordinate;
    this.placedBomb = placedBomb;
    this.health = health;
    this.baseHealth = baseHealth;
    this.speed = speed;
```

```
        this.baseSpeed = baseSpeed;
        this.size = size;
        this.deathCounter = deathCounter;
    }

    public boolean getState(PlayerInfo org) {
        if(id == org.getId())
        {
            org.setCoordinate(coordinate);
            org.setPlacedBomb(placedBomb);
            org.setHealth(health);
            org.setBaseHealth(baseHealth);
            org.setSpeed(baseSpeed);
            org.setBaseSpeed(baseSpeed);
            org.setSize(size);

            return true;
        }
        return false;

    }

}
```

```
package server;

import java.util.*;


```

```
public class Caretaker {

  ArrayList<Memento> statesList;

  public Caretaker(){
    statesList = new ArrayList<Memento>();
  }

  public void add(Memento state){
    statesList.add(state);
  }

  public Memento get(int index){
    Memento restoreState = statesList.get(index);
    statesList.remove(index);
    return restoreState;
  }

  public Memento undo()
  {
    //popping last state
    int index = statesList.size() -1;
    Memento restoreState = statesList.get(index);
    statesList.remove(index);
    return restoreState;
  }


  public int size(){
        return statesList.size();
    }

}
```

```
package server;

import shared.PacketUpdatePlayerPos;
import shared.Vector2f;

import java.util.Random;

public class PlayerInfo
{
    int id;
    Vector2f coordinate;
    boolean isHoldingLeft;
    boolean isHoldingRight;
    boolean isHoldingUp;
    boolean isHoldingDown;
    boolean isHoldingUse;
    boolean isHoldingSkill;
    boolean isHoldingPause;
    boolean isHoldingUnPause;
    boolean placedBomb = true;
    int health;
    int baseHealth;
    float speed;
    float baseSpeed;
```

```
int size;
int bombCount;
int bombTimer;

int deathCounter;

SkillAlgorithm skillAlgorithm;

boolean canDie = true;

public PlayerInfo()
{

  deathCounter = 0;

  Random r = new Random();
  int low = 100;
  int high = 800;

  setBombTimer(1);



  int x = r.nextInt(high-low) + low;
  int y = r.nextInt(high-low) + low;

  this.baseHealth = 2;
  this.coordinate = new Vector2f();
  this.coordinate.x = x;
```

```
    this.coordinate.y = y;
    this.isHoldingLeft = false;
    this.isHoldingRight = false;
    this.isHoldingUp = false;
    this.isHoldingDown = false;
    this.isHoldingPause = false;
    this.isHoldingUnPause = false;
    this.skillAlgorithm = new GoingBackInTimeSkill();
    this.size = 40;
    this.speed = 5f;
    this.baseSpeed = 5f;
    this.health = 2;
    this.bombCount = 2;
    //this.playerStats = new ConcretePlayer();
  }

  public void setBombTimer(int time){
    bombTimer = 30*time;
  }


  public void reduceTimer(){
    bombTimer--;
  }


  public void ChangePlayerLocation(int x, int y){

    this.coordinate.x = x;
    this.coordinate.y = y;

  }

  public PlayerInfo(int id, Vector2f coordinate)
  {
    this.id = id;
    this.coordinate = coordinate;
    this.isHoldingLeft = false;
    this.isHoldingRight = false;
    this.isHoldingUp = false;
    this.isHoldingDown = false;
    this.isHoldingUse = false;
    this.isHoldingPause = false;
    this.isHoldingUnPause = false;
    this.size = 40;
    this.speed = 2.5f;
    this.baseSpeed = 2.5f;
    this.health = 3;
    this.bombCount = 2;
    //Test
  }

  public SkillAlgorithm getSkillAlgorithm()
  {
    return skillAlgorithm;
  }

  public void setSkillAlgorithm(SkillAlgorithm skillAlgorithm)
```

```java
{
    this.skillAlgorithm = skillAlgorithm;
}


public void onTick()
{
    this.skillAlgorithm.onTick(this);
}

public void tryUsingSpell()
{
    this.skillAlgorithm.useSkill(this);
}

public int getCooldown()
{
    return this.skillAlgorithm.getCooldown();
}

public String getName()
{
    return this.skillAlgorithm.getName();
}

public PacketUpdatePlayerPos getPacketUpdatePlayerPos(int id)
{
    PacketUpdatePlayerPos newPack = new PacketUpdatePlayerPos();
    newPack.id = id;
```

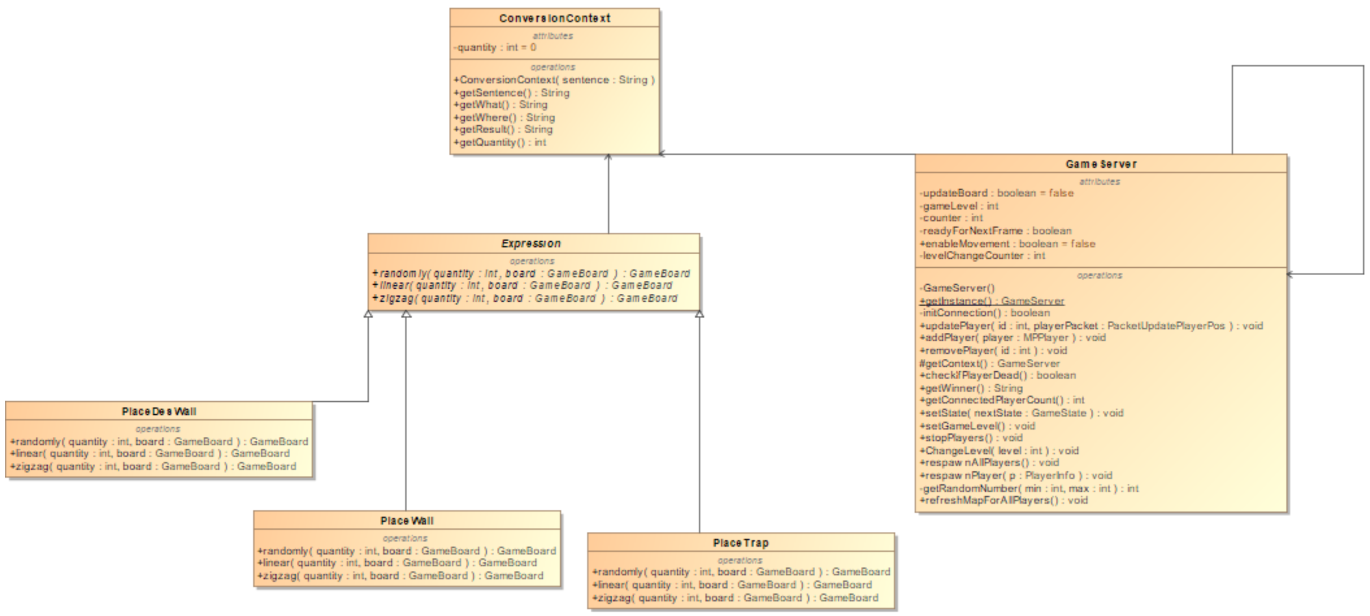
```
        newPack.coordinate = this.coordinate;
        newPack.isHoldingLeft = this.isHoldingLeft;
        newPack.isHoldingRight = this.isHoldingRight;
        newPack.isHoldingUp = this.isHoldingUp;
        newPack.isHoldingDown = this.isHoldingDown;
        newPack.isHoldingUse = this.isHoldingUse;
        newPack.size = this.size;
        newPack.isHoldingSkill = this.isHoldingSkill;
        newPack.isHoldingPause = this.isHoldingPause;
        newPack.isHoldingUnPause = this.isHoldingUnPause;
        newPack.skillName = this.getName();
        newPack.skillCooldown = this.getCooldown();
        newPack.health = this.health;
        return newPack;
    }

    public void restoreState(Memento memento){

        if(memento.getState(this)){
            System.out.println("Successfully restored state");
        }else{
            System.out.println("Unable to restore for Caretaker " + this.id);
        }

    }

    public Memento saveState(){
        return new Memento(id, coordinate, placedBomb, health, baseHealth,
     speed, baseSpeed, size, bombCount, bombTimer, deathCounter);
```

```
}

public int getId() {
  return id;
}

public void setId(int id) {
  this.id = id;
}

public Vector2f getCoordinate() {
  return coordinate;
}

public void setCoordinate(Vector2f coordinate) {
  this.coordinate = new Vector2f(coordinate.x, coordinate.y);
}

public boolean isPlacedBomb() {
  return placedBomb;
}

public void setPlacedBomb(boolean placedBomb) {
  this.placedBomb = placedBomb;
}

public int getHealth() {
  return health;
}
```

```

public void setHealth(int health) {
  this.health = health;
}

public int getBaseHealth() {
  return baseHealth;
}

public void setBaseHealth(int baseHealth) {
  this.baseHealth = baseHealth;
}

public float getSpeed() {
  return speed;
}

public void setSpeed(float speed) {
  this.speed = speed;
}

public float getBaseSpeed() {
  return baseSpeed;
}

public void setBaseSpeed(float baseSpeed) {
  this.baseSpeed = baseSpeed;
}

```

```
  public int getSize() {
    return size;
  }

  public void setSize(int size) {
    this.size = size;
  }

  public int getBombCount() {
    return bombCount;
  }

  public void setBombCount(int bombCount) {
    this.bombCount = bombCount;
  }

  public int getDeathCounter() {
    return deathCounter;
  }

  public void setDeathCounter(int deathCounter) {
    this.deathCounter = deathCounter;
  }

  public int getBombTimer() {
    return bombTimer;
  }

}
```

Interpreter


ConversionContext
attributes
-quantity : int = 0
operations
+ConversionContext(sentence : String)
+getSentence() : String
+getWhat() : String
+getWhere() : String
+getResult() : String
+getQuantity() : int
Expression
operations
+randomly(quantity : int, board : GameBoard) : GameBoard
+linear(quantity : int, board : GameBoard) : GameBoard
+zigzag(quantity : int, board : GameBoard) : GameBoard
GameServer
attributes
-updateBoard : boolean = false
-gameLevel : int
-counter : int
-readyForNextFrame : boolean
+enableMovement : boolean = false
-levelChangeCounter : int
operations
-GameServer()
+getInstance() : GameServer
-initConnection() : boolean
+updatePlayer(id : int, playerPacket : PacketUpdatePlayerPos) : void
+addPlayer(player : MPPlayer) : void
+removePlayer(id : int) : void
#getContext() : GameServer
+checkIfPlayerDead() : boolean
+getWinner() : String
+getConnectedPlayerCount() : int
+setState(nextState : GameState) : void
+setGameLevel() : void
+stopPlayers() : void
+ChangeLevel(level : int) : void
+respawnAllPlayers() : void
+respawnPlayer(p : PlayerInfo) : void
-getRandomNumber(min : int, max : int) : int
+refreshMapForAllPlayers() : void
PlaceDesWall
operations
+randomly(quantity : int, board : GameBoard) : GameBoard
+linear(quantity : int, board : GameBoard) : GameBoard
+zigzag(quantity : int, board : GameBoard) : GameBoard
PlaceWall
operations
+randomly(quantity : int, board : GameBoard) : GameBoard
+linear(quantity : int, board : GameBoard) : GameBoard
+zigzag(quantity : int, board : GameBoard) : GameBoard
PlaceTrap
operations
+randomly(quantity : int, board : GameBoard) : GameBoard
+linear(quantity : int, board : GameBoard) : GameBoard
+zigzag(quantity : int, board : GameBoard) : GameBoard

Šis šablonas buno naudingas pagyvinti žaidimą naudojant console komandas

Interpreter

```java
package server;

import java.util.Random;

public class PlaceDesWall extends Expression{
    @Override
    public GameBoard randomly(int quantity, GameBoard board) {

        Random r = new Random();
        int low = 1;
        int high = 19;


        for (int i = 0; i<quantity; i++){
            int x = r.nextInt(high-low) + low;
            int y = r.nextInt(high-low) + low;

            board.addObject(x, y, "desWall");

        }


        return board;
    }

    @Override
    public GameBoard linear(int quantity, GameBoard board) {
```

28

```


        for (int i = 0; i<quantity; i++){

            int x = 9;
            int y = 9 + i;




            board.addObject(x, y, "desWall");

        }


        return board;
    }

    @Override
    public GameBoard zigzag(int quantity, GameBoard board) {


        int kiekis = quantity;





        for (int i = 1; i<20; i++){

            int x = i;
            int y=i;


            if(kiekis != 0){

                kiekis --;
            }else
            {
                board.addObject(x, y, "desWall");
            }



        }


        return board;
    }
}
```

```
package server;

import java.util.Random;

public class PlaceWall extends Expression{


```

```java
    @Override
    public GameBoard randomly(int quantity, GameBoard board) {

        Random r = new Random();
        int low = 1;
        int high = 19;


        for (int i = 0; i<quantity; i++){
            int x = r.nextInt(high-low) + low;
            int y = r.nextInt(high-low) + low;


            board.addObject(x, y, "wall");


        }


        return board;
    }

    @Override
    public GameBoard linear(int quantity, GameBoard board) {





        for (int i = 0; i<quantity; i++){

            int x = 9 + i;
            int y = 9;




            board.addObject(x, y, "wall");

        }


        return board;
    }

    @Override
    public GameBoard zigzag(int quantity, GameBoard board) {


        for (int i = 0; i<quantity; i++){

            int x = 8 + i;
            int y = 8 + i;



            board.addObject(x, y, "wall");

```

```java

        }


        return board;
    }
}
```

```java
package server;

import java.util.Random;

public class PlaceTrap extends Expression{
    @Override
    public GameBoard randomly(int quantity, GameBoard board) {

        Random r = new Random();
        int low = 1;
        int high = 19;



        for (int i = 0; i<quantity; i++){
            int x = r.nextInt(high-low) + low;
            int y = r.nextInt(high-low) + low;

            board.addObject(x, y, "trap");

        }



        return board;
    }

    @Override
    public GameBoard linear(int quantity, GameBoard board) {


        for (int i = 0; i<quantity; i++){

            int x = 4 + i*2;
            int y = 9;




            board.addObject(x, y, "trap");

        }


        return board;
    }

    @Override
    public GameBoard zigzag(int quantity, GameBoard board) {

        int reverse = -1;
```

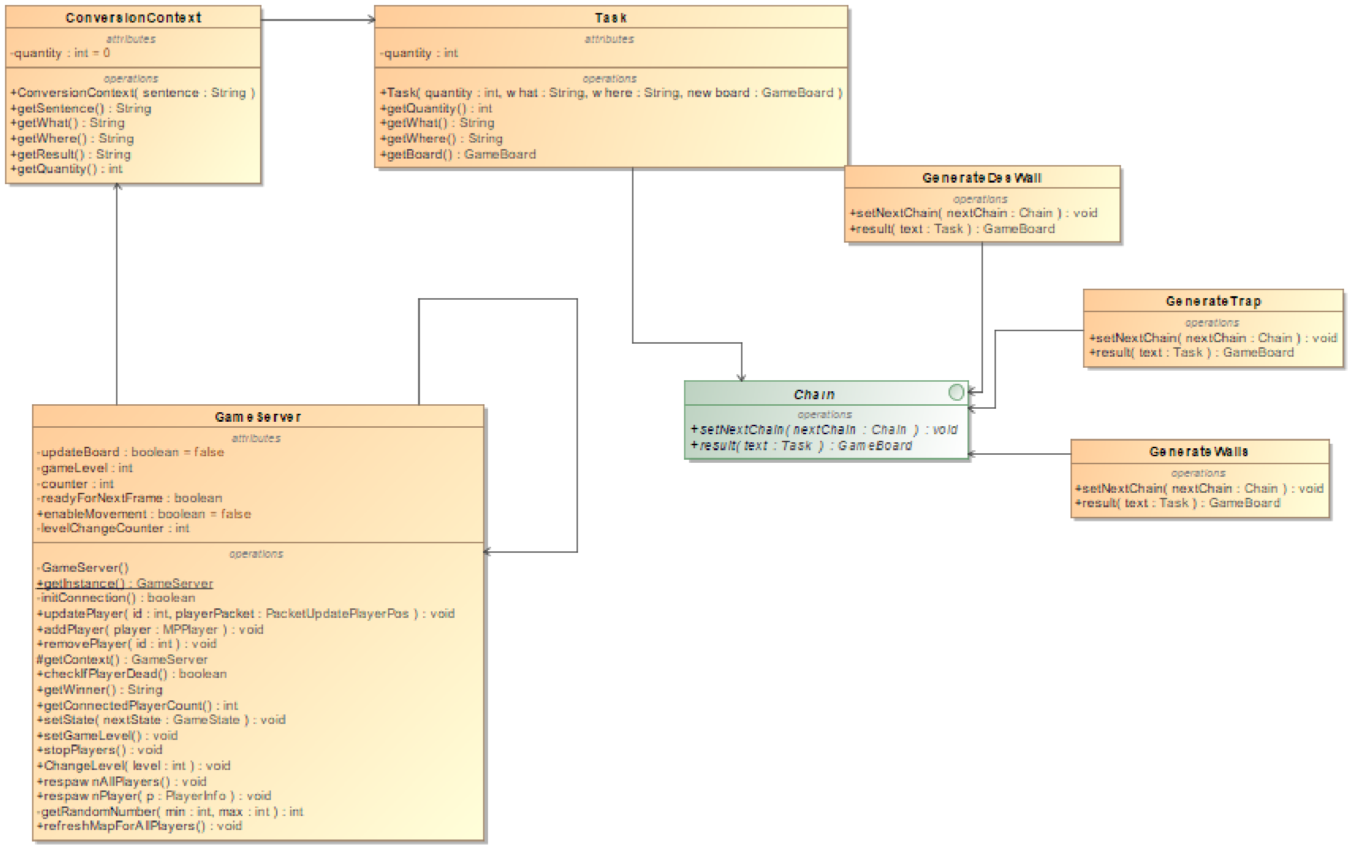
```java

        for (int i = 0; i<quantity; i++){

            int x = 4 + i;
            int y = 9 + (i * reverse);

            reverse *=-1;



            board.addObject(x, y, "trap");

        }


        return board;
    }
}
```

```java
package server;

public abstract class Expression {

    public abstract GameBoard randomly(int quantity, GameBoard board);
    public abstract GameBoard linear(int quantity, GameBoard board);
    public abstract GameBoard zigzag(int quantity, GameBoard board);



}
```

```
package server;

import java.util.Locale;

public class ConversionContext {

    private String givenSentence = "";

    private String result = "";

    private String what = "";
    private String where = "";

    private int quantity = 0;

    String[] parts;

    public ConversionContext(String sentence){

        givenSentence = sentence.toLowerCase();

        parts = getSentence().split(" ");

        what = parts[2];

        quantity = Integer.parseInt(parts[1]);

```

```
            where = parts[3];

            result = givenSentence + " ";

        }

        public String getSentence(){
            return givenSentence;
        }

        public String getWhat(){
            return what;
        }

        public String getWhere(){
            return where;
        }

        public String getResult(){
            return result;
        }

        public int getQuantity(){
            return  quantity;
        }



}
```

Chain of responsibility


ConversionContext
attributes
-quantity : int = 0
operations
+ConversionContext(sentence : String)
+getSentence() : String
+getWhat() : String
+getWhere() : String
+getResult() : String
+getQuantity() : int
Task
attributes
-quantity : int
operations
+Task(quantity : int, what : String, where : String, new board : GameBoard)
+getQuantity() : int
+getWhat() : String
+getWhere() : String
+getBoard() : GameBoard
GenerateDesWall
operations
+setNextChain(nextChain : Chain) : void
+result(text : Task) : GameBoard
GenerateTrap
operations
+setNextChain(nextChain : Chain) : void
+result(text : Task) : GameBoard
Chain
operations
+setNextChain(nextChain : Chain) : void
+result(text : Task) : GameBoard
GenerateWalls
operations
+setNextChain(nextChain : Chain) : void
+result(text : Task) : GameBoard
GameServer
attributes
-updateBoard : boolean = false
-gameLevel : int
-counter : int
-readyForNextFrame : boolean
+enableMovement : boolean = false
-levelChangeCounter : int
operations
-GameServer()
+getInstance() : GameServer
-initConnection() : boolean
+updatePlayer(id : int, playerPacket : PacketUpdatePlayerPos) : void
+addPlayer(player : MPPlayer) : void
+removePlayer(id : int) : void
#getContext() : GameServer
+checkIfPlayerDead() : boolean
+getWinner() : String
+getConnectedPlayerCount() : int
+setState(nextState : GameState) : void
+setGameLevel() : void
+stopPlayers() : void
+ChangeLevel(level : int) : void
+respawnAllPlayers() : void
+respawnPlayer(p : PlayerInfo) : void
-getRandomNumber(min : int, max : int) : int
+refreshMapForAllPlayers() : void

Šį šabloną pasirinkau, kad pagyvinti interpreter darbą

Chain of responsibility

```
package server;

public class Task {
    private int quantity;
    private GameBoard board;
    private String what;
    private String where;

    public Task(int quantity, String what, String where, GameBoard
  newboard){
        this.quantity = quantity;
        this.what = what;
        this.where = where;
        board = newboard;
    }



    public int getQuantity(){
        return quantity;
    }

    public String getWhat(){
        return what;
    }

    public String getWhere(){
        return where ;
```

```
    }

    public GameBoard getBoard(){
        return board;
    }
}
```

```
package server;

public interface Chain {

     public void setNextChain(Chain nextChain);
    public GameBoard result(Task text);
}
```

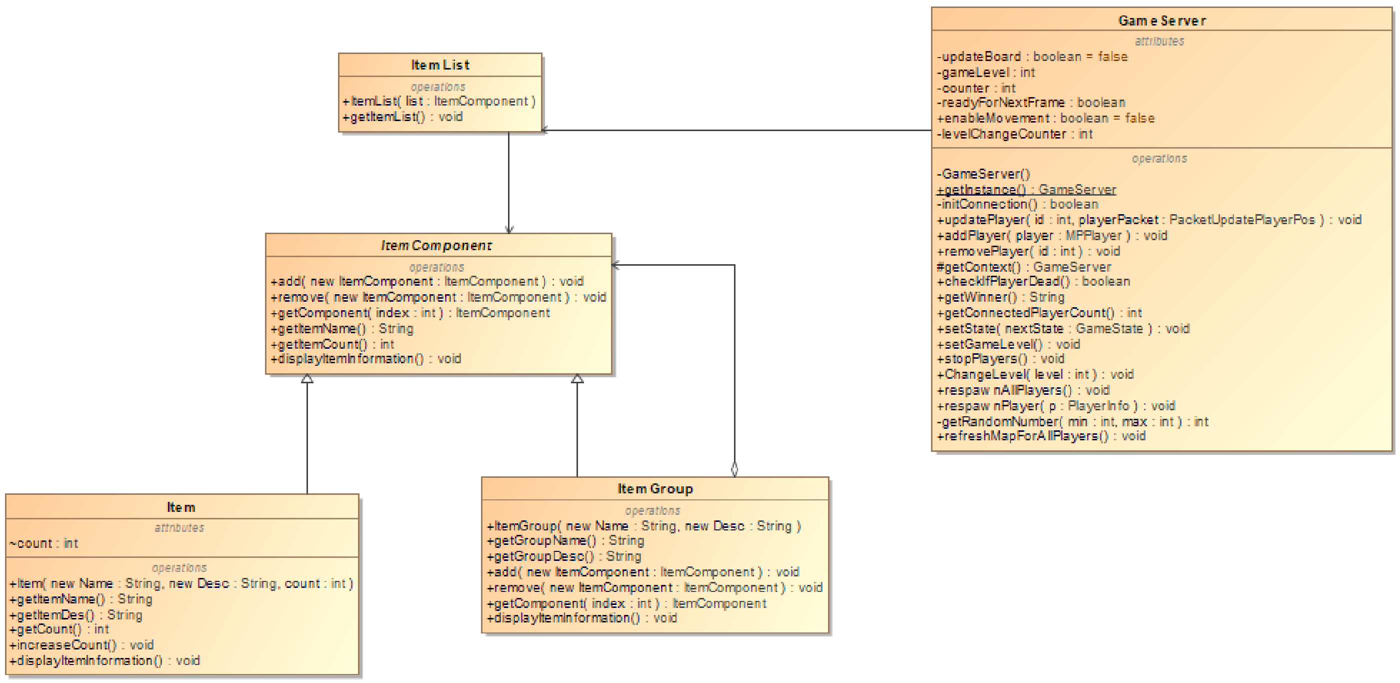
```
package server;

public class GenerateWalls implements Chain {



    private Chain nextInChain;

    @Override
     public void setNextChain(Chain nextChain) {
        nextInChain =nextChain;

    }

    @Override
    public GameBoard result(Task text) {

        GameBoard result = text.getBoard();

        if(text.getWhat().equals("wall")){

            PlaceWall newWalls = new PlaceWall();

            System.out.println("Turiu sienas sukurti ");

            switch (text.getWhere()){
                case "randomly": result=newWalls.randomly(text.
    getQuantity(),text.getBoard());
                    break;

                case "zigzag": result =newWalls.zigzag(text.
    getQuantity(),text.getBoard());
                    break;

                case "linear": result=newWalls.linear(text.
    getQuantity(),text.getBoard());
                    break;

            }

            return result;



        }else{
                nextInChain.result(text);
            }



            return result;

        }
}
```

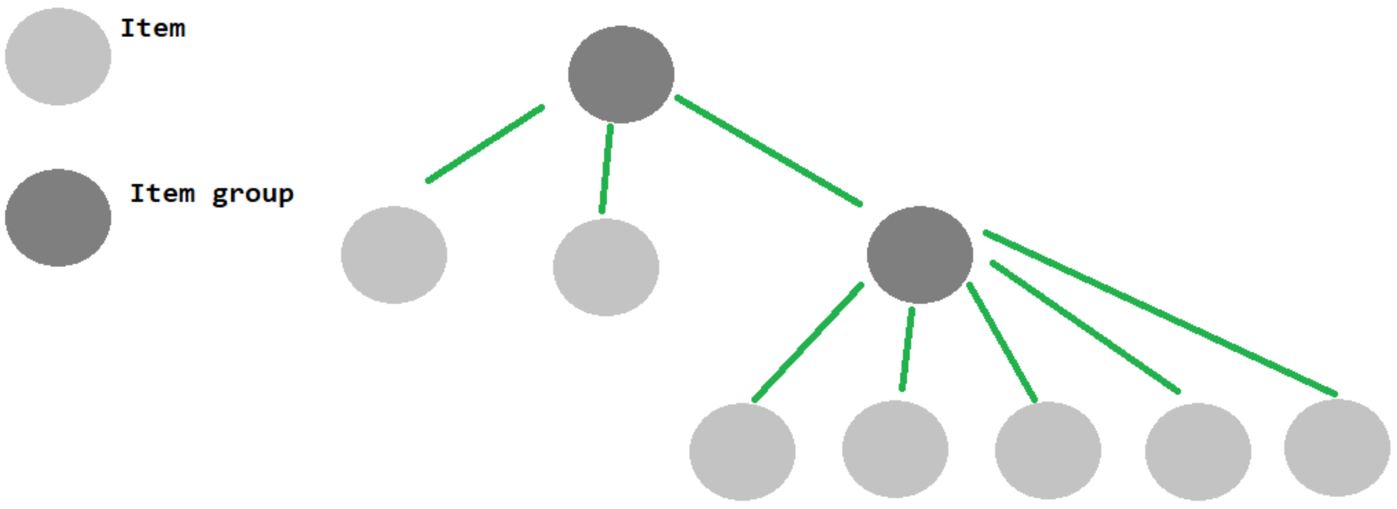
```
package server;

public class GenerateTrap implements Chain {
        private Chain nextInChain;

        @Override
        public void setNextChain(Chain nextChain) {
            nextInChain =nextChain;

        }

        @Override
        public GameBoard result(Task text) {

            GameBoard result = text.getBoard();

            if(text.getWhat().equals("trap")){

                PlaceTrap newWalls = new PlaceTrap();
            switch (text.getWhere()){
                case "randomly": result=newWalls.randomly(text.
getQuantity(),text.getBoard());
                    break;

                case "zigzag": result = newWalls.zigzag(text.
getQuantity(),text.getBoard());
                    break;

                case "linear": result=newWalls.linear(text.
getQuantity(),text.getBoard());
                    break;

            }

            return result;



        }else{
            nextInChain.result(text);
        }



        return result;

    }
```

```

}
```

```
package server;

public class GenerateDesWall implements Chain {
    private Chain nextInChain;

    @Override
    public void setNextChain(Chain nextChain) {
        nextInChain = nextChain;

    }

    @Override
    public GameBoard result(Task text) {

        GameBoard result = text.getBoard();

        if(text.getWhat().equals("deswall")){

            PlaceDesWall newWalls = new PlaceDesWall();

            switch (text.getWhere()){
                case "randomly": result= newWalls.randomly(text.
    getQuantity(),text.getBoard());
                    break;

                case "zigzag": result = newWalls.zigzag(text.
getQuantity(), text.getBoard());
                    break;

                case "linear": result=newWalls.linear(text.
getQuantity(),text.getBoard());
                    break;

            }

            return result;



        }



        return result;

    }
}
```

Composite


Item List
operations
+ItemList(list : ItemComponent)
+getItemList() : void
Item Component
operations
+add(new ItemComponent : ItemComponent) : void
+remove(new ItemComponent : ItemComponent) : void
+getComponent(index : int) : ItemComponent
+getItemName() : String
+getItemCount() : int
+displayItemInformation() : void
Item
attributes
~count : int
operations
+Item(new Name : String, new Desc : String, count : int)
+getItemName() : String
+getItemDes() : String
+getCount() : int
+increaseCount() : void
+displayItemInformation() : void
Item Group
operations
+ItemGroup(new Name : String, new Desc : String)
+getGroupName() : String
+getGroupDesc() : String
+add(new ItemComponent : ItemComponent) : void
+remove(new ItemComponent : ItemComponent) : void
+getComponent(index : int) : ItemComponent
+displayItemInformation() : void
Game Server
attributes
-updateBoard : boolean = false
-gameLevel : int
-counter : int
-readyForNextFrame : boolean
+enableMovement : boolean = false
-levelChangeCounter : int
operations
-GameServer()
+getInstance() : GameServer
-initConnection() : boolean
+updatePlayer(id : int, playerPacket : PacketUpdatePlayerPos) : void
+addPlayer(player : MPPlayer) : void
+removePlayer(id : int) : void
#getContext() : GameServer
+checkIfPlayerDead() : boolean
+getWinner() : String
+getConnectedPlayerCount() : int
+setState(nextState : GameState) : void
+setGameLevel() : void
+stopPlayers() : void
+ChangeLevel(level : int) : void
+respawnAllPlayers() : void
+respawnPlayer(p : PlayerInfo) : void
-getRandomNumber(min : int, max : int) : int
+refreshMapForAllPlayers() : void



Šį algoritmą pasirinkau, nes reikėjo būdo surinkti informaciją apie žaidimą ir ją pavaizduoti.

Composition

```
package server;

public class Item extends ItemComponent{

    String itemName;
    String itemDes;
```

```
  int count;

  public Item(String newName, String newDesc, int count){
      itemName = newName;
      itemDes = newDesc;
      this.count = count;
  }


  public String getItemName(){
      return itemName;
  }

  public String getItemDes(){
      return itemDes;
  }

  public int getCount(){
      return count;
  }

  public void increaseCount(){
      count++;
  }

  public void displayItemInformation(){
      System.out.println(itemName + " Desc:  " + itemDes + " Count:  "
+ count);
  }
```

```
}
```

```java
package server;

public abstract class ItemComponent {

    public void add(ItemComponent newItemComponent){
        throw new UnsupportedOperationException();
    }

    public void remove(ItemComponent newItemComponent){
        throw new UnsupportedOperationException();
    }

    public ItemComponent getComponent(int index){
        throw new UnsupportedOperationException();
    }

    public String getItemName(){
        throw new UnsupportedOperationException();
    }

    public int getItemCount(){
        throw new UnsupportedOperationException();
    }

    public void displayItemInformation(){
        throw new UnsupportedOperationException();
    }



}
```

```java
package server;


import java.util.ArrayList;
import java.util.Iterator;

public class ItemGroup extends ItemComponent{

    ArrayList itemComponents = new ArrayList();

    String groupName;
    String groupDesc;

    public ItemGroup(String newName, String newDesc){
        groupName = newName;
        groupDesc = newDesc;
    }


    public String getGroupName(){
        return groupName;
    }

    public String getGroupDesc(){
        return groupDesc;
    }

    public void add(ItemComponent newItemComponent){
        itemComponents.add(newItemComponent);
    }

    public void remove(ItemComponent newItemComponent){
        itemComponents.remove(newItemComponent);
    }

    public ItemComponent getComponent(int index){

        return (ItemComponent) itemComponents.get(index);
    }


    public void displayItemInformation(){
        System.out.println(getGroupName() + "\n" + getGroupDesc() + "\
n");


        Iterator itemIterator = itemComponents.iterator();

        while (itemIterator.hasNext()){
            ItemComponent itemInfo = (ItemComponent) itemIterator.next
();
            itemInfo.displayItemInformation();
        }

        System.out.println();
        System.out.println();
    }





}
```

```java
package server;

public class ItemList {

    ItemComponent itemList;

    public ItemList(ItemComponent list){
        itemList = list;
    }

    public void getItemList(){
        itemList.displayItemInformation();
    }

}
```