**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №5**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: Добавление врагов**

Студент гр. 9381 _____ Колованов Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

**Цель работы.**

Изучить парадигму объектно-ориентрированного программирования; реализовать шаблонный класс врагов; изучить и реализовать паттерн проектирования *State*.

**Задание.**

Создать шаблонный класс врага. Параметр шаблона должен определять поведение врага *(параметров шаблона может быть несколько, например отдельный параметр для политики передвижения и для политики атаки)*. Класс врага должен препятствовать игроку. Класс игрока должен иметь возможность взаимодействовать с врагом и наоборот.

**Обязательные требования:**

- Создан шаблонный класс врага;
- Создано не менее 3 типа поведения врагов;
- Взаимодействие происходит через перегруженный оператор.

**Дополнительные требования:**

- Передача хода между игроком и врагами происходит с использованием паттерна **Состояния** в классе игры.

**Выполнение работы.**

Для начала были реализован шаблонный класс врага *Enemy*. В качестве параметров шаблона принимаются классы политики перемещения: *StandMovementBehavior* и *WalkMovementBehavior,* и классы политики атаки *MeleeAttackBehavior* и *DistanceAttackBehavior* соответственно. У базового класса Creature (производными от которого являются классы *Player* и *Enemy*) был реализован оператор <=, при помощи которого классы живых сущностей могут атаковать друг друга. По мимо этого, был реализован паттерн *Состояние* для класса *GameController. Были* реализованы интрефейс для состояний *GameState* и сами классы состояний *PlayerTurnState* и *EnemiesTurnState,* при помощи которых происходит передача хода между игроком и врагами.

В программе используются умные указатели, поэтому очистка памяти для них не требуется. Для реализации GUI-интерфейса программы был использован фреймворк *Qt*.

Подробное описание классов приведено ниже (см. Раздел *Описание классов, структур и перечислений*).

**Описание классов, структур и перечислений.**

***Класс* Abstract*Enemy***.

Является наследником класса *Creature*. Представляет собой интерфейс для классов врагов.

Методы класса Abstract*Enemy*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *void* | *operator<=(sharedObject& object) override = 0* |
| *public* | *void* | *operator<=(sharedCreature& creature) override = 0* |
| *public* | *Position2D* | *getMovementPosition(const Position2D& target_position) = 0* |
| *public* | *void* | *tryAttack(sharedCreature& target) = 0* |
| *public* | *const std::type_info&* | *getClass() const = 0* |

***Класс Enemy***.

Шаблонный класс, является наследником класса *AbstractEnemy*. Представляет собой врага. Шаблон принимает два класса: класс политики передвижения и класс политики атаки врага соответственно.

Методы класса *Enemy*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | - | *explicit Enemy(Position2D position)* |
| *public* | - | *~Enemy() override* |
| *public* | *void* | *operator<=(sharedObject& object) override* |
| *public* | *void* | *operator<=(sharedCreature& creature) override* |

| | | |
|---|---|---|
| *public* | *Position2D* | *getMovementPosition(const Position2D& target_position) override* |
| *public* | *void* | *tryAttack(sharedCreature& target) override* |
| *public* | *const std::type_info&* | *getClass() const override* |

***Класс GameController.***

Представляет собой класс для управления игрой, помимо этого является фасадом, работающим с подсистемой классов игры. Контролирует переход хода между игроком и врагами.

Поля класса *GameController*:

| Модификатор доступа | Название и тип поля | Предназначение | Значение по умолчанию |
|---|---|---|---|
| *private* | *pPlayer player_* | Хранит адрес объекта игрока. | - |
| *private* | *Enemies enemies_* | Хранит вектор указателей на врагов. | - |
| *private* | pLoggingListener loggingListener_ | Хранит адрес объекта для отслеживания игровых объектов и вывод логов на консоль и в файл. | - |
| *private* | sharedGameState state_ | Хранит текущее состояние контроллера. | - |
| *private* | size_t level_ | Хранит информацию о текущем уровне игры. | *0* |
| *private* | bool levelComplete_ | Хранит информацию о том, завершил ли игрок текущий уровень. | *false* |

Методы класса *GameController*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | - | *GameController(pLoggingListener& logger)* |
| *public* | - | *~GameController()* |
| *public* | *void* | *startTurn()* |
| *public* | *void* | *endTurn()* |
| *public* | *void* | *createLevel()* |
| *public* | *void* | *movePlayer(Direction direction)* |
| *public* | *void* | *executePlayerInteraction()* |
| *public* | *void* | *executePlayerAttack()* |
| *public* | *void* | *changeState(const sharedGameState& state)* |
| *public* | *void* | *checkLevelFinish()* |
| *public* | *bool* | *isLevelComplete()* |
| *public* | *size_t* | *getLevelNumber() const* |
| *public* | *bool* | *isPlayerReachedExit() const* |
| *public* | *bool* | *isPassablePosition(const Position2D& position) const* |
| *public* | *void* | *getLevelPixmap(sharedQPixmap& level_pixmap) const* |
| *public* | *sharedPlayer* | *getPlayer()* |
| *public* | *Enemies&* | *getEnemies()* |

***Класс GameState.***

Представляет собой интерфейс для классов состояний контроллера.

Методы класса *GameState*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *void* | *startTurn(GameController& controller)* |

| | | |
|---|---|---|
| | | *= 0* |
| *public* | *void* | *endTurn(GameController& controller) = 0* |
| *public* | *void* | *movePlayer(GameController& controller, Direction direction) = 0* |
| *public* | *void* | *executePlayerInteraction(GameController& controller) = 0* |
| *public* | *void* | *executePlayerAttack(GameController& controller) = 0* |

***Класс PlayerTurnState.***

Представляет собой состояние, при котором ходит игрок.

Методы класса *PlayerTurnState*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *void* | *startTurn(GameController& controller) override* |
| *public* | *void* | *endTurn(GameController& controller) override* |
| *public* | *void* | *movePlayer(GameController& controller, Direction direction) override* |
| *public* | *void* | *executePlayerInteraction(GameController& controller) override* |
| *public* | *void* | *executePlayerAttack(GameController& controller) override* |

***Класс EnemiesTurnState***.

Представляет собой состояние, при котором ходят враги.

Методы класса *EnemiesTurnState*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *void* | *startTurn(GameController& controller) override* |
| *public* | *void* | *endTurn(GameController& controller) override* |
| *public* | *void* | *movePlayer(GameController& controller, Direction direction) override* |
| *public* | *void* | *executePlayerInteraction(GameController& controller) override* |
| *public* | *void* | *executePlayerAttack(GameController& controller) override* |

***Класс AttackBehavior.***

Представляет собой интерфейс для классов политики атаки врагов.

Методы класса *AttackBehavior*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *void* | *attack(Creature& creature, sharedCreature& target) = 0* |

***Класс MeleeAttackBehavior.***

Представляет собой классов политики ближнего боя врагов.

Методы класса *MeleeAttackBehavior*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *void* | *attack(Creature& creature, sharedCreature& target) override* |

***Класс DistanceAttackBehavior.***

Представляет собой классов политики дальнего боя врагов.

Методы класса *DistanceAttackBehavior*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *void* | *attack(Creature& creature, sharedCreature& target) override* |

***Класс MovementBehavior.***

Представляет собой интерфейс для классов политики передвижения врагов.

Методы класса *AttackBehavior*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *Position2D* | *getMovementPosition(Creature& creature, const Position2D& target_position) = 0* |

***Класс StandMovementBehavior.***

Представляет собой класс политики передвижения врагов: не двигаться.

Методы класса StandMovement*Behavior*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *Position2D* | *getMovementPosition(Creature& creature, const Position2D&* |

| | | |
|---|---|---|
| | | *target_position) override* |

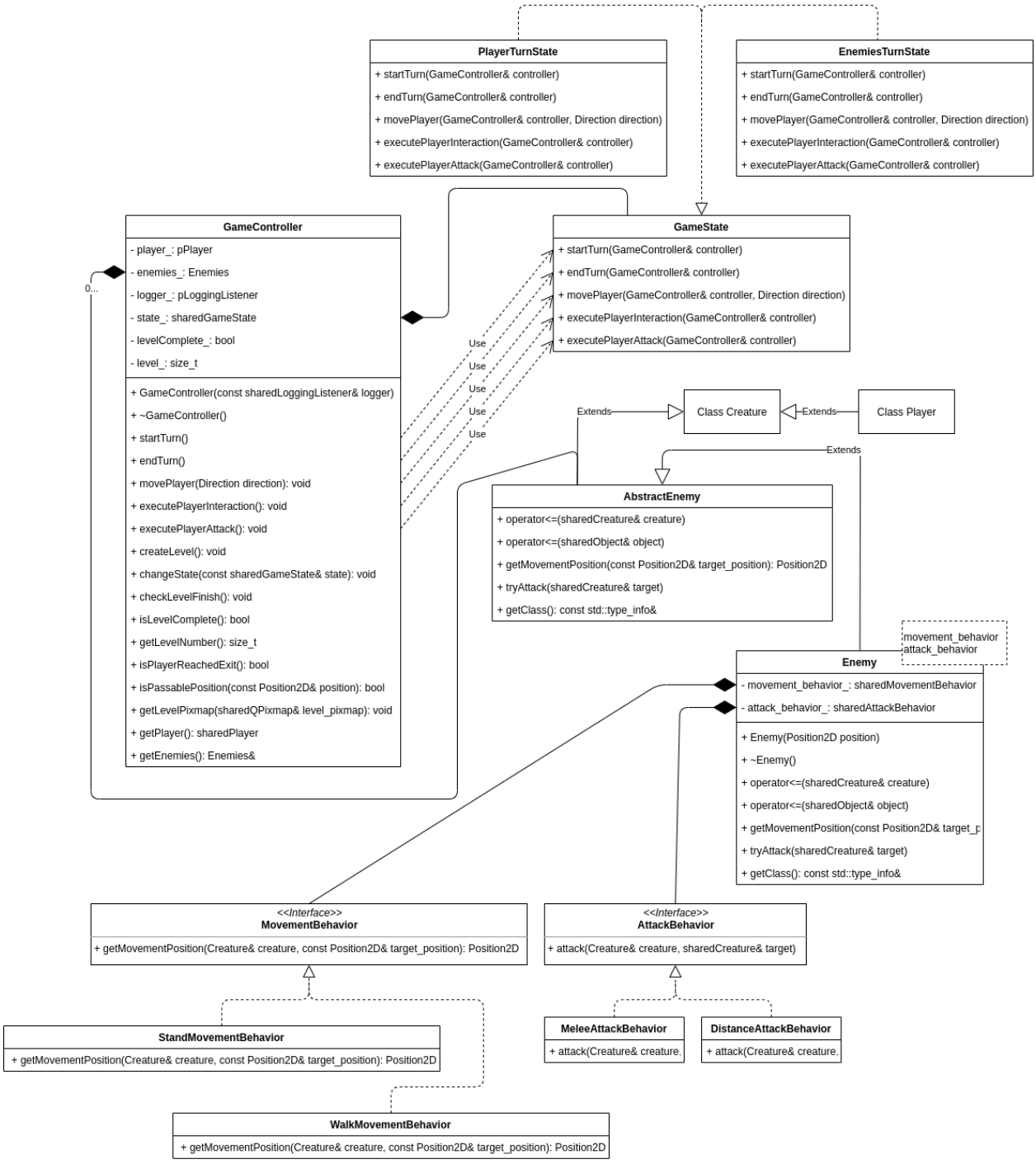
***Класс WalkMovementBehavior.***

Представляет собой класс политики передвижения врагов: двигаться на одну клетку за ход.

Методы класса *WalkMovementBehavior*:

| Модификатор доступа | Возвращаемое значение | Название метода и принимаемые аргументы |
|---|---|---|
| *public* | *Position2D* | *getMovementPosition(Creature& creature, const Position2D& target_position) override* |

# UML-диаграмма.



**Тестирование.**

**Файл log.txt**

```
[01-12-20 02:32:35] Object of class 'Player' change position to
[2, 2]
[01-12-20 02:32:41] Object of class 'Player' change position to
[2, 3]
[01-12-20 02:32:41] Object of class 'Player' change position to
[2, 4]
[01-12-20 02:32:41] Object of class 'Player' change position to
[2, 5]
[01-12-20 02:32:41] Object of class 'Player' change position to
[2, 6]
[01-12-20 02:32:42] Object of class 'Player' change position to
[2, 7]
[01-12-20 02:32:42] Object of class 'Player' change position to
[2, 8]
[01-12-20 02:32:42] Object of class 'Enemy' attack creature of
class '6Player'
[01-12-20 02:32:42] Object of class 'Player' change health to 99
[01-12-20 02:32:42] Object of class 'Player' change position to
[2, 9]
[01-12-20 02:32:42] Object of class 'Enemy' attack creature of
class '6Player'
[01-12-20 02:32:42] Object of class 'Player' change health to 98
[01-12-20 02:32:42] Object of class 'Player' change position to
[2, 10]
[01-12-20 02:32:42] Object of class 'Enemy' attack creature of
class '6Player'
[01-12-20 02:32:42] Object of class 'Player' change health to 97
[01-12-20 02:32:43] Object of class 'Player' attack creature of
class '5EnemyIL14MovementPolicy0EL12AttackPolicy1EE'
[01-12-20 02:32:43] Object of class 'Enemy' attack creature of
class '6Player'
[01-12-20 02:32:43] Object of class 'Player' change health to 96
[01-12-20 02:32:43] Object of class 'Player' attack creature of
class '5EnemyIL14MovementPolicy0EL12AttackPolicy1EE'
[01-12-20 02:32:43] Object of class 'Enemy' attack creature of
class '6Player'
[01-12-20 02:32:43] Object of class 'Player' change health to 95
[01-12-20 02:32:44] Object of class 'Player' attack creature of
class '5EnemyIL14MovementPolicy0EL12AttackPolicy1EE'
[01-12-20 02:32:44] Object of class 'Enemy' attack creature of
class '6Player'
[01-12-20 02:32:44] Object of class 'Player' change health to 94
[01-12-20 02:32:44] Object of class 'Player' attack creature of
class '5EnemyIL14MovementPolicy0EL12AttackPolicy1EE'
[01-12-20 02:32:44] Object of class 'Enemy' attack creature of
class '6Player'
[01-12-20 02:32:44] Object of class 'Player' change health to 93
[01-12-20 02:32:44] Object of class 'Player' attack creature of
class '5EnemyIL14MovementPolicy0EL12AttackPolicy1EE'
[01-12-20 02:32:44] Destroying: Object of class 'Enemy'.
[01-12-20 02:32:46] Quitting the game...
```

```
     [01-12-20 02:32:46] Destroying: Object of class 'Enemy'.
     [01-12-20 02:32:46] Destroying: Object of class 'Enemy'.
     [01-12-20 02:32:46] Destroying: Object of class 'Enemy'.
     [01-12-20 02:32:46] Destroying: Object of class 'Player':
Position((2, 10)); Health(93); MaxHealth(100); AttackDamage(2);
Protection(0); Rotation(3); PassFounded(0)
     [01-12-20 02:32:46] Destroying: Object of class 'Armor':
ProtectionValue(1)
     [01-12-20 02:32:46] Destroying: Object of class 'Medicines':
HealthRecovery(20)
     [01-12-20 02:32:46] Destroying: Object of class
'LevelPassObject'.
     [01-12-20 02:32:46] Destroying: Object of class 'Armor':
ProtectionValue(1)
     [01-12-20 02:32:46] Destroying: Object of class 'Armor':
ProtectionValue(1)
     [01-12-20 02:32:46] Destroying: Object of class 'Armor':
ProtectionValue(1)
```

**Выводы.**

Была изучена парадигма объектно-ориентрированного программирования. Были реализованы шаблонный класс врага *Enemy*, интерфейс *GameState* для состояний хода и его наследники: *PlayerTurnState* и *EnemiesTurnState*. Были изучен и реализован паттерн проектирования *State*. По мимо этого, был реализован GUI-интерфейс игры при помощи фреймворка Qt.

# ПРИЛОЖЕНИЕ А

# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: GameController.h

```
#ifndef SOURCES_GAME_GAMECONTROLLER_H
#define SOURCES_GAME_GAMECONTROLLER_H

#include <QPixmap>

#include "sources/game/objects/creatures/player/player.h"
#include "sources/game/objects/creatures/enemies/enemy.h"
#include "sources/game/field.h"
#include "sources/game/gamestate.h"
#include "sources/logging/logginglistener.h"

using sharedGameController = std::shared_ptr<class GameController>;
using sharedQPixmap = std::shared_ptr<QPixmap>;
using Enemies = std::vector<sharedCreature>;
using sharedGameState = std::shared_ptr<class GameState>;


class GameController final {
private:
    sharedPlayer player_;
    Enemies enemies_;
    sharedLoggingListener logger_;
    sharedGameState state_;
    size_t level_ = 0;
    bool level_complete_ = false;

public:
    GameController(const sharedLoggingListener& logger);
    ~GameController();

    void startTurn();
    void endTurn();
    void createLevel();
    void movePlayer(Direction direction);
    void executePlayerInteraction();
    void executePlayerAttack();
    void changeState(const sharedGameState& state);
    void checkLevelFinish();
    bool isLevelComplete();
    size_t getLevelNumber() const;
    bool isPlayerReachedExit() const;
    bool isPassablePosition(const Position2D& position) const;
    void getLevelPixmap(sharedQPixmap& level_pixmap) const;
    sharedPlayer getPlayer();
    Enemies& getEnemies();
}; // class GameController


#endif // SOURCES_GAME_GAMECONTROLLER_H
```

Название файла: GameController.cpp

```
#include <QMap>
```

```
#include "sources/game/gamecontroller.h"
#include "sources/game/levelgenerator.h"
#include "sources/game/objects/creatures/enemies/enemy.h"
#include "sources/gui/levelpainter.h"
#include "sources/game/playerturnstate.h"


GameController::GameController(const sharedLoggingListener& logger):
logger_(logger) {
    changeState(std::make_shared<PlayerTurnState>());
    player_ = std::make_shared<Player>(Position2D(0, 0));
    player_->setMaxHealth(100);
    player_->setHealth(100);
    player_->setAttackDamage(2);
    player_->setProtection(0);
    player_->getEventManager().subscribe(logger);
}


void GameController::createLevel() {
    LevelGenerator levelGenerator(logger_);
    levelGenerator.generate(Size2D(2 + level_, 2 + level_), level_);
    enemies_.clear();
    levelGenerator.spawnEnemies(enemies_, level_);
    level_complete_ = false;
    player_->setPosition(levelGenerator.getEntryPosition());
    player_->setPassFounded(false);
}


void GameController::changeState(const sharedGameState& state) {
    state_ = state;
}


void GameController::startTurn() {
    if (state_ != nullptr) {
        state_->startTurn(*this);
    }
}


void GameController::endTurn() {
    if (state_ != nullptr) {
        state_->endTurn(*this);
    }
}


void GameController::movePlayer(Direction direction) {
    if (state_ != nullptr) {
        state_->movePlayer(*this, direction);
    }
}


void GameController::executePlayerInteraction() {
    if (state_ != nullptr) {
        state_->executePlayerInteraction(*this);
    }
}
```

```
      void GameController::executePlayerAttack() {
          if (state_ != nullptr) {
              state_->executePlayerAttack(*this);
          }
      }


      void GameController::checkLevelFinish() {
          if (isPlayerReachedExit() && player_->getPassFounded() && !
level_complete_) {
              level_complete_ = true;
              level_++;
          }
      }


      bool GameController::isLevelComplete() {
          return level_complete_;
      }


      bool GameController::isPlayerReachedExit() const {
          const Field& field = Field::getInstance();
          return field.getCell(player_->getPosition()).getType() ==
CellType::Exit;
      }


      bool GameController::isPassablePosition(const Position2D& position)
const {
          const Field& field = Field::getInstance();
          if (field.getCell(position).isPassable()) {
              for (auto& enemy : enemies_) {
                  if (enemy->getPosition() == position) {
                      return false;
                  }
              }
              return position != player_->getPosition();
          }
          return false;
      }


      void GameController::getLevelPixmap(sharedQPixmap& levelPixmap) const {
          gui::LevelPainter::paint(levelPixmap, player_, enemies_);
      }


      sharedPlayer GameController::getPlayer() {
          return player_;
      }


      Enemies& GameController::getEnemies() {
          return enemies_;
      }


      size_t GameController::getLevelNumber() const {
```

```
        return level_;
    }


    GameController::~GameController() {
        logger_->update("Quitting the game...");
    }
```

Название файла: Enemy.h

```
    #ifndef SOURCES_GAME_OBJECTS_CREATURES_ENEMIES_ENEMY_H
    #define SOURCES_GAME_OBJECTS_CREATURES_ENEMIES_ENEMY_H

    #include <memory>

    #include "sources/game/objects/creatures/enemies/abstractenemy.h"
    #include "sources/game/objects/creatures/attackbehavior.h"
    #include "sources/game/objects/creatures/movementbehavior.h"
    #include "sources/game/interactions/interactionnone.h"


    template<typename movement_behavior, typename attack_behavior>
    class Enemy final: public AbstractEnemy {
    private:
        sharedMovementBehavior movement_behavior_;
        sharedAttackBehavior attack_behavior_;

    public:
        explicit Enemy(Position2D position);
        ~Enemy() override;

        void operator<=(sharedObject& object) override;
        void operator<=(sharedCreature& creature) override;

        Position2D getMovementPosition(const Position2D& target_position)
override;
        void tryAttack(sharedCreature& target) override;
        const std::type_info& getClass() const override;
    }; // class Enemy


    template<typename movement_behavior, typename attack_behavior>
    using sharedEnemy = std::shared_ptr<class Enemy<movement_behavior,
attack_behavior>>;


    template<typename movement_behavior, typename attack_behavior>
    Enemy<movement_behavior, attack_behavior>::Enemy(Position2D position) {
        setInteractionStrategy(std::make_shared<InteractionNone>());
        setPosition(position);
        movement_behavior_ = std::make_shared<movement_behavior>();
        attack_behavior_ = std::make_shared<attack_behavior>();
    }


    template<typename movement_behavior, typename attack_behavior>
    Enemy<movement_behavior, attack_behavior>::~Enemy() {
        getEventManager().notify("Destroying: Object of class 'Enemy'.");
    }
```

```
    template<typename movement_behavior, typename attack_behavior>
    void Enemy<movement_behavior, attack_behavior>::operator<=(sharedObject&
object) {
        if (object_interaction_strategy_ != nullptr) {
            if (object != nullptr) {
                std::ostringstream ss;
                ss << "Object of class 'Enemy' interact with object of class
'" << object->getClass().name() << "'";
                getEventManager().notify(ss);
            }
            object_interaction_strategy_->interact(*this, object);
        }
    }


    template<typename movement_behavior, typename attack_behavior>
    void Enemy<movement_behavior,
attack_behavior>::operator<=(sharedCreature& creature) {
        if (creature != nullptr) {
            std::ostringstream ss;
            ss << "Object of class 'Enemy' attack creature of class '" <<
creature->getClass().name() << "'";
            getEventManager().notify(ss);
            Creature::operator<=(creature);
        }
    }


    template<typename movement_behavior, typename attack_behavior>
    Position2D Enemy<movement_behavior,
attack_behavior>::getMovementPosition(const Position2D& target_position) {
        return movement_behavior_->getMovementPosition(*this,
target_position);
    }


    template<typename movement_behavior, typename attack_behavior>
    void Enemy<movement_behavior,
attack_behavior>::tryAttack(sharedCreature& target) {
        attack_behavior_->attack(*this, target);
    }


    template<typename movement_behavior, typename attack_behavior>
    const std::type_info& Enemy<movement_behavior,
attack_behavior>::getClass() const {
        return typeid(Enemy);
    }


    #endif // SOURCES_GAME_OBJECTS_CREATURES_ENEMIES_ENEMY_H
```

Название файла: GameState.h

```
    #ifndef SOURCES_GAME_GAME_STATE_H
    #define SOURCES_GAME_GAME_STATE_H

    #include "sources/game/gamecontroller.h"
```

```
using sharedGameController = std::shared_ptr<class GameController>;
using sharedGameState = std::shared_ptr<class GameState>;


class GameState {
public:
    virtual void startTurn(GameController& controller) = 0;
    virtual void endTurn(GameController& controller) = 0;
    virtual void movePlayer(GameController& controller, Direction
direction) = 0;
    virtual void executePlayerInteraction(GameController& controller) =
0;
    virtual void executePlayerAttack(GameController& controller) = 0;
}; // class GameState


#endif // SOURCES_GAME_GAME_STATE_H
```

Название файла: PlayerTurnState.h

```
#ifndef SOURCES_GAME_PLAYER_TURN_STATE_H
#define SOURCES_GAME_PLAYER_TURN_STATE_H

#include "sources/game/gamestate.h"

using sharedPlayerTurnState = std::shared_ptr<class PlayerTurnState>;


class PlayerTurnState final: public GameState {
public:
    virtual void startTurn(GameController& controller) override;
    virtual void endTurn(GameController& controller) override;
    virtual void movePlayer(GameController& controller, Direction
direction) override;
    virtual void executePlayerInteraction(GameController& controller)
override;
    virtual void executePlayerAttack(GameController& controller)
override;
}; // class PlayerTurnState


#endif // SOURCES_GAME_PLAYER_TURN_STATE_H
```

Название файла: PlayerTurnState.cpp

```
#include "sources/game/playerturnstate.h"
#include "sources/game/enemiesturnstate.h"


void PlayerTurnState::startTurn(GameController&) {}


void PlayerTurnState::endTurn(GameController& controller) {
    controller.checkLevelFinish();
    controller.changeState(std::make_shared<EnemiesTurnState>());
    controller.startTurn();
}
```

```
      void PlayerTurnState::movePlayer(GameController& controller, Direction
direction) {
          sharedPlayer player = controller.getPlayer();
          Position2D new_position = player->getPosition();

          new_position.shift(direction);
          player->setRotation(direction);

          if (controller.isPassablePosition(new_position)) {
              player->setPosition(new_position);
          }

          controller.endTurn();
      }


      void PlayerTurnState::executePlayerInteraction(GameController&
controller) {
          Field& field = Field::getInstance();
          sharedPlayer player = controller.getPlayer();
          Position2D interaction_position = player-
>getPosition().shift(player->getRotation());
          *player <= field.getCell(interaction_position).getObject();

          controller.endTurn();
      }

      void PlayerTurnState::executePlayerAttack(GameController& controller) {
          sharedPlayer player = controller.getPlayer();
          Position2D interaction_position = player-
>getPosition().shift(player->getRotation());
          sharedCreature attack_enemy = nullptr;
          size_t enemy_index = 0;

          for (auto& enemy : controller.getEnemies()) {
              if (enemy->getPosition() == interaction_position) {
                  attack_enemy = enemy;
                  break;
              }
              enemy_index++;
          }

          *player <= attack_enemy;

          if (attack_enemy != nullptr && attack_enemy->getHealth() <= 0) {
              controller.getEnemies().erase(controller.getEnemies().begin() +
enemy_index);
          }

          controller.endTurn();
      }
```

Название файла: EnemiesTurnState.h

```
      #ifndef SOURCES_GAME_ENEMIES_TURN_STATE_H
      #define SOURCES_GAME_ENEMIES_TURN_STATE_H

      #include "sources/game/gamestate.h"
```

```
using sharedEnemiesTurnState = std::shared_ptr<class EnemiesTurnState>;


class EnemiesTurnState final: public GameState {
public:
    virtual void startTurn(GameController& controller) override;
    virtual void endTurn(GameController& controller) override;
    virtual void movePlayer(GameController& controller, Direction
direction) override;
    virtual void executePlayerInteraction(GameController& controller)
override;
    virtual void executePlayerAttack(GameController& controller)
override;
}; // class EnemiesTurnState


#endif // SOURCES_GAME_ENEMIES_TURN_STATE_H
```

Название файла: EnemiesTurnState.cpp

```
#include "sources/game/enemiesturnstate.h"
#include "sources/game/playerturnstate.h"


void EnemiesTurnState::startTurn(GameController& controller) {
    sharedCreature player = controller.getPlayer();

    for (auto& enemy : controller.getEnemies()) {
        Position2D player_position = player->getPosition();
        Position2D enemy_position = enemy->getPosition();

        long long delta_x = static_cast<long long>(player_position.x) -
static_cast<long long>(enemy_position.x);
        long long delta_y = static_cast<long long>(player_position.y) -
static_cast<long long>(enemy_position.y);

        if (abs(delta_x) < 7 && abs(delta_y) < 7) {
            Position2D new_position = enemy_position;

            // Movement
            if (enemy->getClass() == typeid(Enemy<MovementPolicy::Walk,
AttackPolicy::Distance>) ||
                enemy->getClass() == typeid(Enemy<MovementPolicy::Walk,
AttackPolicy::Melee>)) {
                if (abs(delta_x) > abs(delta_y)) {
                    if (delta_x > 0) {
                        new_position.shift(Direction::Right);
                        enemy->setRotation(Direction::Right);
                    } else if (delta_x < 0) {
                        new_position.shift(Direction::Left);
                        enemy->setRotation(Direction::Left);
                    } else if (delta_y > 0) {
                        new_position.shift(Direction::Bottom);
                        enemy->setRotation(Direction::Bottom);
                    } else if (delta_y < 0) {
                        new_position.shift(Direction::Top);
                        enemy->setRotation(Direction::Top);
                    }
                } else {
```

```
                        if (delta_y > 0) {
                            new_position.shift(Direction::Bottom);
                            enemy->setRotation(Direction::Bottom);
                        } else if (delta_y < 0) {
                            new_position.shift(Direction::Top);
                            enemy->setRotation(Direction::Top);
                        } else if (delta_x > 0) {
                            new_position.shift(Direction::Right);
                            enemy->setRotation(Direction::Right);
                        } else if (delta_x < 0) {
                            new_position.shift(Direction::Left);
                            enemy->setRotation(Direction::Left);
                        }
                    }
                }

                if (enemy_position != new_position &&
controller.isPassablePosition(new_position)) {
                    enemy->setPosition(new_position);
                    enemy_position = new_position;
                    delta_x = static_cast<long long>(player_position.x) -
static_cast<long long>(enemy_position.x);
                    delta_y = static_cast<long long>(player_position.y) -
static_cast<long long>(enemy_position.y);
                }

                // Attack
                if (enemy->getClass() == typeid(Enemy<MovementPolicy::Stand,
AttackPolicy::Melee>) ||
                    enemy->getClass() == typeid(Enemy<MovementPolicy::Walk,
AttackPolicy::Melee>)) {
                    if (Position2D(enemy_position.x + 1, enemy_position.y)
== player_position) {
                        enemy->setRotation(Direction::Right);
                        *enemy <= player;
                    } else if (Position2D(enemy_position.x - 1,
enemy_position.y) == player_position) {
                        enemy->setRotation(Direction::Left);
                        *enemy <= player;
                    } else if (Position2D(enemy_position.x, enemy_position.y
+ 1) == player_position) {
                        enemy->setRotation(Direction::Bottom);
                        *enemy <= player;
                    } else if (Position2D(enemy_position.x, enemy_position.y
- 1) == player_position) {
                        enemy->setRotation(Direction::Top);
                        *enemy <= player;
                    }
                } else {
                    if (delta_x > 0 && delta_y == 0 && abs(delta_x) < 4) {
                        enemy->setRotation(Direction::Right);
                        *enemy <= player;
                    } else if (delta_x < 0 && delta_y == 0 && abs(delta_x) <
4) {
                        enemy->setRotation(Direction::Left);
                        *enemy <= player;
                    } else if (delta_y > 0 && delta_x == 0 && abs(delta_y) <
4) {
                        enemy->setRotation(Direction::Bottom);
                        *enemy <= player;
```

```
                    } else if (delta_y < 0 && delta_x == 0 && abs(delta_y) <
4) {
                        enemy->setRotation(Direction::Top);
                        *enemy <= player;
                    }
                }
            }
        }

        controller.endTurn();
    }


    void EnemiesTurnState::endTurn(GameController& controller) {
        controller.changeState(std::make_shared<PlayerTurnState>());
        controller.startTurn();
    }


    void EnemiesTurnState::movePlayer(GameController&, Direction) {}


    void EnemiesTurnState::executePlayerInteraction(GameController&) {}


    void EnemiesTurnState::executePlayerAttack(GameController&) {}
```

Название файла: AbstractEnemy.h

```
    #ifndef SOURCES_GAME_OBJECTS_CREATURES_ENEMIES_ABSTRACT_ENEMY_H
    #define SOURCES_GAME_OBJECTS_CREATURES_ENEMIES_ABSTRACT_ENEMY_H

    #include <memory>
    #include <vector>

    #include "sources/game/objects/creatures/creature.h"
    #include "sources/game/interactions/interactionnone.h"

    using sharedAbstractEnemy = std::shared_ptr<class AbstractEnemy>;
    using Enemies = std::vector<sharedAbstractEnemy>;


    class AbstractEnemy: public Creature {
    public:
        void operator<=(sharedObject& object) override = 0;
        void operator<=(sharedCreature& creature) override = 0;
        virtual Position2D getMovementPosition(const Position2D&
target_position) = 0;
        virtual void tryAttack(sharedCreature& target) = 0;
        const std::type_info& getClass() const override = 0;
    }; // class AbstractEnem


    #endif // SOURCES_GAME_OBJECTS_CREATURES_ENEMIES_ABSTRACT_ENEMY_H
```