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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно ориентированное программирование»

Тема: ИНТЕРФЕЙСЫ, ДИНАМИЧЕСКИЙ ПОЛИМОРФИЗМ

Студент гр. 1304 Шаврин А.П.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Изучить интерфейсы и динамический полиморфизм в ООП в языке C++ и применить полученные знания в реализации игры. Научиться создавать классы интерфейсы, а также выстраивать архитектуру проекта.

Задание.

Реализовать систему событий. Событие - сущность, которая срабатывает при взаимодействии с игроком. Должен быть разработан класс интерфейс общий для всех событий, поддерживающий взаимодействие с игроком. Необходимо создать несколько групп разных событий реализуя унаследованные от интерфейса события (например, враг, который проверяет условие, будет ли воздействовать на игрока или нет; ловушка, которая безусловно воздействует на игрока; событие , которое меняет карту; и.т.д.). Для каждой группы реализовать конкретные события, которые по разному воздействуют на игрока (например, какое-то событие заставляет передвинуться игрока в определенную сторону, а другое меняет характеристики игрока). Также, необходимо предусмотреть событие “Победа/Выход”, которое срабатывает при соблюдении определенного набора условий.
Реализовать ситуацию проигрыша (например, потери всего здоровья игрока) и выигрыша игрока (добрался и активировал событие “Победа/Выход”)

Требования:

- Разработан интерфейс события с необходимым описанием методов
- Реализовано минимум 2 группы событий (2 абстрактных класса наследников события)
- Для каждой группы реализовано минимум 2 конкретных события (наследники от группы события)
- Реализовано минимум одно условное и безусловное событие (условное - проверяет выполнение условий, безусловное - не проверяет).
- Реализовано минимум одно событие, которое меняет карту (меняет события на клетках или открывает расположение выхода или делает какие-то клетки проходимыми (на них необходимо добавить события) или не непроходимыми
- Игрок в гарантированно имеет возможность дойти до выхода

Примечания:

- Классы событий не должны хранить никакой информации о типе события (никаких переменных и функций дающие информации о типе события)
- Для создания события можно применять абстрактную фабрику/прототип/строитель

Выполнение работы.

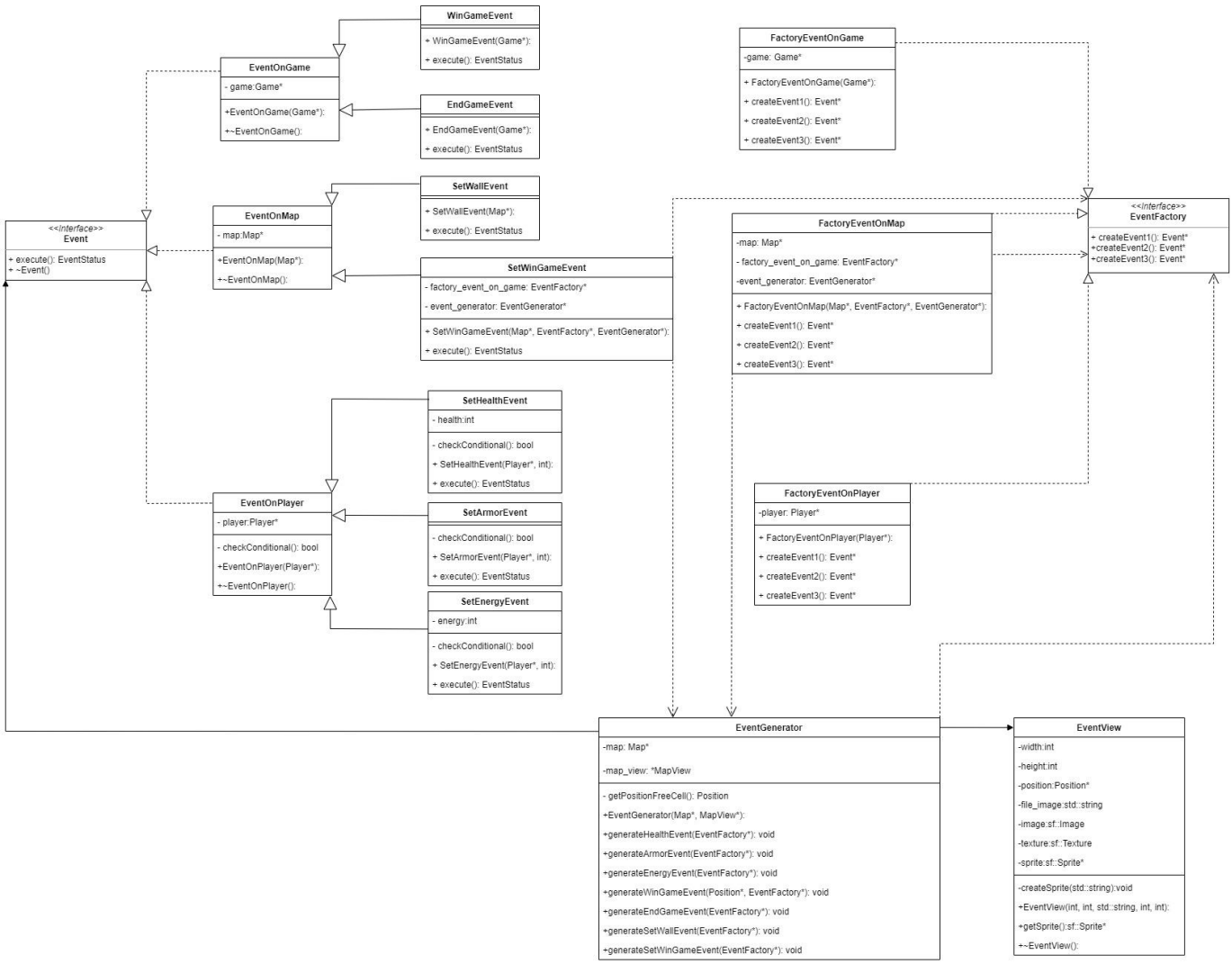
1. Было создано перечисление EventStatus из 2х элементов: DELETE – информирующее о том, что событие отработало и его нужно удалить; LEAVE - информирующее о том, что событие не отработало и его не нужно удалять.
2. Был создан класс интерфейс Event, с виртуальным методом execute, который возвращает элемент перечисления EventStatus.
3. Создан абстрактный класс EventOnGame, унаследованный от класса Event, объединяющий несколько событий в категорию событий, которые воздействуют на игру. В себе он хранит одно поле: указатель на класс Game, для уведомления этого класса о состоянии игры в событиях.
4. Было создано событие WinGameEvent, унаследованное от абстрактного класса EventOnGame. Данное событие в переопределенном методе execute изменяет в классе Game информацию о состоянии игры, выводит сообщение о победе, возвращает DELETE.
5. Было создано событие EndGameEvent, унаследованное от абстрактного класса EventOnGame. Данное событие в переопределенном методе execute изменяет в классе Game информацию о состоянии игры, выводит сообщение о проигрыше, возвращает DELETE.
6. Создан абстрактный класс EventOnPlayer, унаследованный от класса Event, объединяющий несколько событий в категорию событий, которые воздействуют на игрока. В себе он хранит одно поле: указатель на класс Player, а также чистую виртуальную функцию checkConditional(), ее переопределят наследуемые события, т.к. они будут условными (проверять условие).
7. Было создано событие SetHealthEvent, унаследованное от абстрактного класса EventOnPlayer. Данное событие в переопределенном методе checkConditional проверяет возможность добавления здоровья, а в методе

execute производит проверку. При прохождении проверки - увеличивает здоровье игрока и возвращает DELETE, иначе – не выполняет начисление игроку здоровья и возвращает LEAVE.

8. Было создано событие SetArmorEvent, унаследованное от абстрактного класса EventOnPlayer. Данное событие в переопределенном методе checkConditional проверяет есть ли у игрока уже броня, а в методе execute производит проверку. При прохождении проверки – устанавливает игроку броню и возвращает DELETE, иначе – не выполняет установку брони и возвращает LEAVE.
9. Было создано событие SetEnergyEvent, унаследованное от абстрактного класса EventOnPlayer. Данное событие в переопределенном методе checkConditional проверяет возможность добавления энергии, а в методе execute производит проверку. При прохождении проверки - увеличивает энергию игрока и возвращает DELETE, иначе – не выполняет начисление игроку энергии и возвращает LEAVE.
10. Создан абстрактный класс EventOnMap, унаследованный от класса Event, объединяющий несколько событий в категорию событий, которые воздействуют на карту. В себе он хранит одно поле: указатель на класс Map.
11. Было создано событие SetWallEvent, унаследованное от абстрактного класса EventOnMap. Данное событие в переопределенном методе execute выбирает рандомную пустую клетку без событий на поле и устанавливает на ней стену и возвращает DELETE
12. Было создано событие SetWinGameEvent, унаследованное от абстрактного класса EventOnMap. Данное событие хранит в себе указатели на фабрику событий и генератор событий, с помощью которых в переопределенном методе execute выбирает рандомную пустую клетку без событий на поле и устанавливает на ней WinGameEvent и возвращает DELETE.
13. Был создан класс-интерфейс EventFactory, имеющий 3 чисто виртуальных метода createEvent1(),createEvent2(),createEvent3(), возвращающие указатель на Event.

14. Создан абстрактный класс FactoryEventOnGame, унаследованный от класса EventFactory, создающий события, влияющие на состояние игры. В себе он хранит одно поле: указатель на класс Game, а переопределяет виртуальные методы. В createEvent1 создается событие победы, в createEvent2 создается событие проигрыша, в createEvent3 не создается ничего.
15. Создан абстрактный класс FactoryEventOnPlayer, унаследованный от класса EventFactory, создающий события, влияющие на состояние игрока. В себе он хранит одно поле: указатель на класс Player, а переопределяет виртуальные методы. В createEvent1 создается событие установки здоровья, в createEvent2 создается событие установки брони, в createEvent3 создается событие установки энергии.
16. Создан абстрактный класс FactoryEventOnMap, унаследованный от класса EventFactory, создающий события, меняющие карту. В себе он хранит поля: указатель на класс Map, указатель на класс EventFactory (для создания события), указатель на класс EventGenerator (для создания и установки события, а также его визуального представления), и переопределяет виртуальные методы. В createEvent1 создается событие установки стены, в createEvent2 создается событие, устанавливающее событие победы, в createEvent3 не создается ничего.
17. Создан класс EventGenerator, с полями: Map* и MapView*. Данный класс имеет методы для создания всех событий. Все методы принимают указатель на интерфейс фабрику событий (метод создания поедного события принимает еще один аргумент - позицию). Во всех методах происходит рандомизация свободной клетки(кроме того, что создает победное событие), на которую будет установлено событие, создается и устанавливается событие на карту, а также его визуальное отображение на внешний вид карты.

Ниже приведена UML диаграмма зависимости классов со всеми их полями и методами:

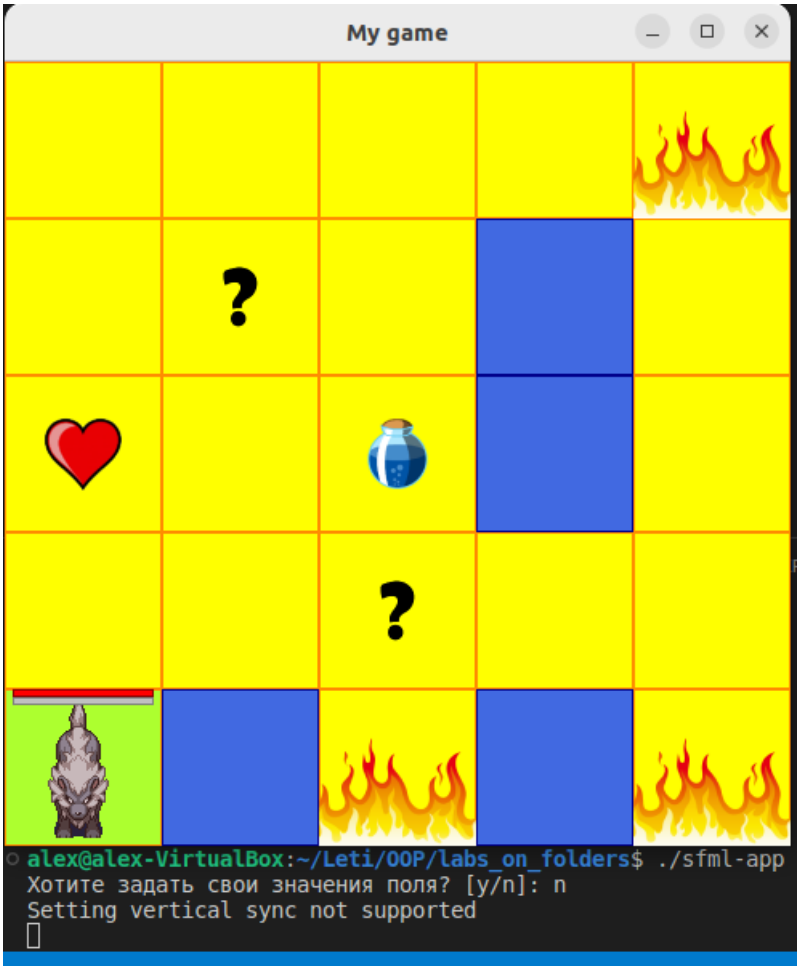

WinGameEvent
+ WinGameEvent(Game*):
+ execute(): EventStatus
EventOnGame
- game:Game*
+EventOnGame(Game*):
+~EventOnGame():
EndGameEvent
+ EndGameEvent(Game*):
+ execute(): EventStatus
FactoryEventOnGame
-game: Game*
+ FactoryEventOnGame(Game*):
+ createEvent1(): Event*
+ createEvent2(): Event*
+ createEvent3(): Event*
SetWallEvent
+ SetWallEvent(Map*):
+ execute(): EventStatus
<<Interface>>
Event
+ execute(): EventStatus
+ ~Event()
EventOnMap
- map:Map*
+EventOnMap(Map*):
+~EventOnMap():
SetWinGameEvent
- factory_event_on_game: EventFactory*
- event_generator: EventGenerator*
+ SetWinGameEvent(Map*, EventFactory*, EventGenerator*):
+ execute(): EventStatus
FactoryEventOnMap
-map: Map*
- factory_event_on_game: EventFactory*
-event_generator: EventGenerator*
+ FactoryEventOnMap(Map*, EventFactory*, EventGenerator*):
+ createEvent1(): Event*
+ createEvent2(): Event*
+ createEvent3(): Event*
<<Interface>>
EventFactory
+ createEvent1(): Event*
+createEvent2(): Event*
+createEvent3(): Event*
SetHealthEvent
- health:int
- checkConditional(): bool
+ SetHealthEvent(Player*, int):
+ execute(): EventStatus
EventOnPlayer
- player:Player*
- checkConditional(): bool
+EventOnPlayer(Player*):
+~EventOnPlayer():
SetArmorEvent
- checkConditional(): bool
+ SetArmorEvent(Player*, int):
+ execute(): EventStatus
SetEnergyEvent
- energy:int
- checkConditional(): bool
+ SetEnergyEvent(Player*, int):
+ execute(): EventStatus
FactoryEventOnPlayer
-player: Player*
+ FactoryEventOnPlayer(Player*):
+ createEvent1(): Event*
+ createEvent2(): Event*
+ createEvent3(): Event*
EventGenerator
-map: Map*
-map_view: *MapView
- getPositionFreeCell(): Position
+EventGenerator(Map*, MapView*):
+generateHealthEvent(EventFactory*): void
+generateArmorEvent(EventFactory*): void
+generateEnergyEvent(EventFactory*): void
+generateWinGameEvent(Position*, EventFactory*): void
+generateEndGameEvent(EventFactory*): void
+generateSetWallEvent(EventFactory*): void
+generateSetWinGameEvent(EventFactory*): void
EventView
-width:int
-height:int
-position:Position*
-file_image:std::string
-image:sf::Image
-texture:sf::Texture
-sprite:sf::Sprite*
-createSprite(std::string):void
+EventView(int, int, std::string, int, int):
+getSprite():sf::Sprite*
+~EventView():

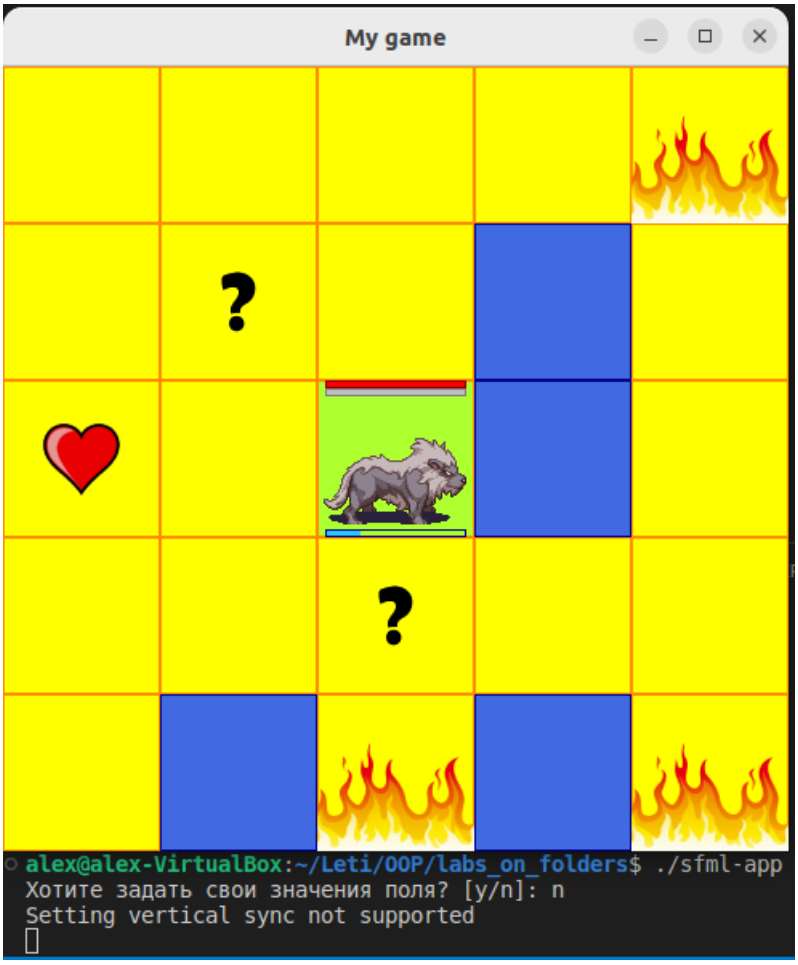
Тестирование.

1. Создание поля с событиями

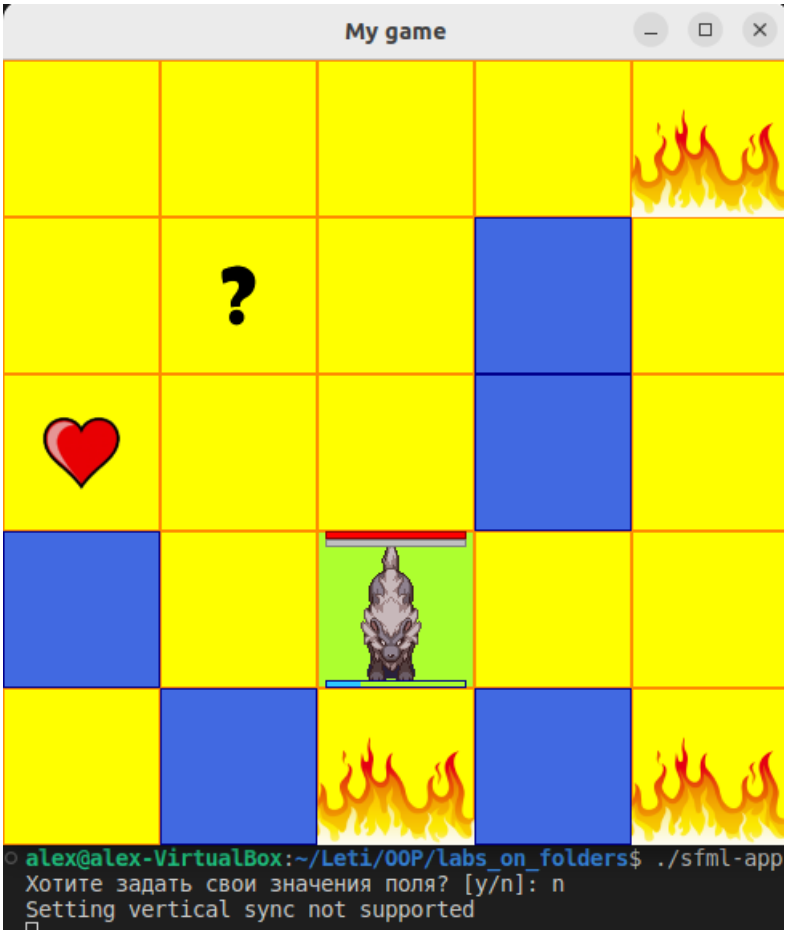
2. Проверка условных событий, влияющих на игрока

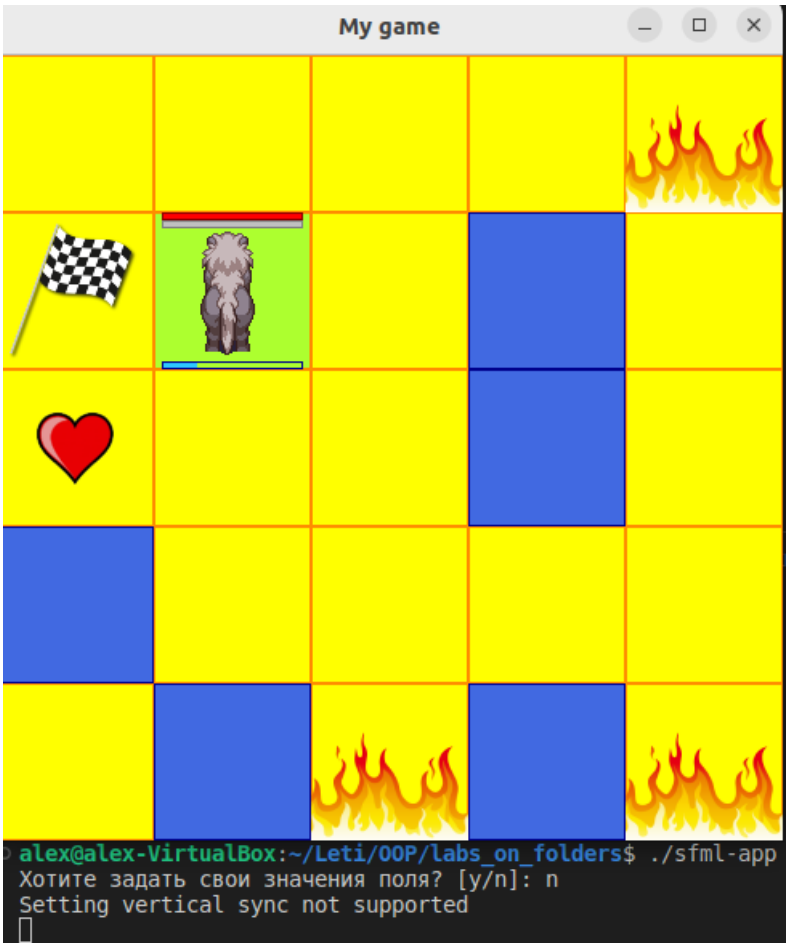

My game
alex@alex-VirtualBox:~/Leti/OOP/labs_on_folders$ ./sfml-app
Хотите задать свои значения поля? [y/n]: n
Setting vertical sync not supported


My game
alex@alex-VirtualBox:~/Leti/OOP/labs_on_folders$ ./sfml-app
Хотите задать свои значения поля? [y/n]: n
Setting vertical sync not supported



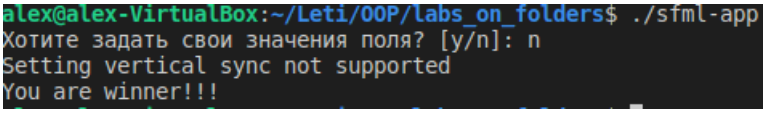
3. Проверка событий, влияющих на карту


My game
alex@alex-VirtualBox:~/Leti/OOP/labs_on_folders$ ./sfml-app
Хотите задать свои значения поля? [y/n]: n
Setting vertical sync not supported

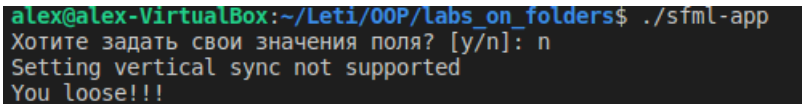

My game
alex@alex-VirtualBox:~/Leti/OOP/labs_on_folders$ ./sfml-app
Хотите задать свои значения поля? [y/n]: n
Setting vertical sync not supported

4. Проверка событий, влияющих на игру

   Игрок встал на клетку с событием победы, игра завершилась.



   Игрок встал на клетку с событием проигрыша, игра завершилась



Выводы.

Были изучены интерфейсы и динамический полиморфизм в ООП в языке C++ и применены полученные знания в реализации игры. Были созданы классы интерфейсы, а также выстроена архитектура проекта.