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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Объектно-ориентированое программирование»
Тема: Интерфейсы, динамический полиморфизм.

Студент гр. 1381 _____ Исайкин Г. И.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2022

Цель работы.

Изучить интерфейс и динамический полиморфизм и использовать их в программировании на языке C++.

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.

Для выполнения работы было сделано множество классов событий:

- CellEvent. Класс является интерфейсом всех событий. Он имеет лишь объявление одного метода — trigger(). Он возвращает GameStatus, объект обозначающий статус игры — победа, проигрыш, убрать событие с клетки или продолжать играть. Этот метод активирует событие у всех объектов, классы которых наследуются напрямую или через другие классы от этого.
- EventPlayerVirtual. Класс интерфейс для всех событий, меняющих игрока, наследник CellEvent. Имеет поле player где хранится указатель на игрока, и методы для его получения и изменения.
- EventCasing. Класс, который вызывает вложенное в него события, хранящиеся в поле CellEvent *event, с определённым шансом и даёт ему имя для опознавания, которое хранится в поле str::string name. Все методы кроме trigger() предназначены для настройки полей или копирования.

- EventResetEvent. Класс, наследник CellEvent. Он хранит указатель на объекта класса SetterCellEvent, через который меняет события на поле в зависимости от настроек самого события, так и от настроек этого объекта.
- EventCombat. Абстрактный класс, который задаёт данные события сражения. То беж все поля и методы существуют для задачи данных. Сами вычисления итогов сражения для игрока высчитываются в методе trigger() в классах наследниках.
- EventCombatHard, EventCombatMedium и EventCombatEasy. Классы наследники EventCombat. У них определён лишь метод trigger(). Отличаются они методом вычисления последствий сражения.
- EventBonus. Событие, которое просто увеличивает или уменьшает характеристики игрока. Все поля и методы (кроме trigger()) отвечают за настройку события.
- EventGetCoins. Класс события, которое увеличивает или уменьшает количество монет у игрока. Также, если поле EventCombat *additional_event хранит не nullptr, параллельно активируется сражение. Все методы (кроме trigger()) и поля отвечают за настройку эффектов события.
- EventGetCoinsHard. Наследник EventGetCoins. Делает тоже самое, но если trigger() не может снять монеты, возвращает lose.
- EventTradeOffer. Класс наследник EventPlayerVirtual. Событие проверяет, достаточно ли у игрока монет, и если да, снимет их, взамен давая преимущества. Все методы (кроме trigger()) и поля отвечают за настройку события.
- EventFinalTradeOffer. Наследник EventPlayerVirtual. Проверяет достаточно ли у игрока денег. Если да, trigger() возвращает win.
- EventFinal. Наследник EventPlayerVirtual. Его trigger() проверяет умение игрока. Если они достаточны, возвращается win. В противном

случаи, если заполнено поле EventCombatHard* boss, вызывает trigger() от EventCombatHard* boss.

- EventFieldVirtual. Аналогичен EventPlayerVirtual, но для событий, изменяющих поле.
- EventChangePassble. Класс наследник от EventFieldVirtual. Он содержит координаты центра области преобразования карты и радиус этой области. Все методы и поля предназначены для настройки. Метод trigger() определён в классах наследниках.
- EventMakeNotPassable. Наследник EventChangePassble. trigger() делает определённую область не проходимой. Также проверяет, что бы после преобразований карты игрок не был зажат.
- EventMakePassable. Наследник EventChangePassble. trigger() делает область проходимой. Также имеет поле bool is_player_coord и методы для его определения, которое при значении true ставят центр области в клетку игрока.
- EventMakeAllPassable Класс наследник от EventFieldVirtual. Его метод trigger() полностью убирает непроходимые клетки на карте, делая их проходимыми.
- EventMakeBorder. Класс наследник от EventFieldVirtual. Его метод trigger() делает по краям поля клетки непроходимыми.

Другие изменения.

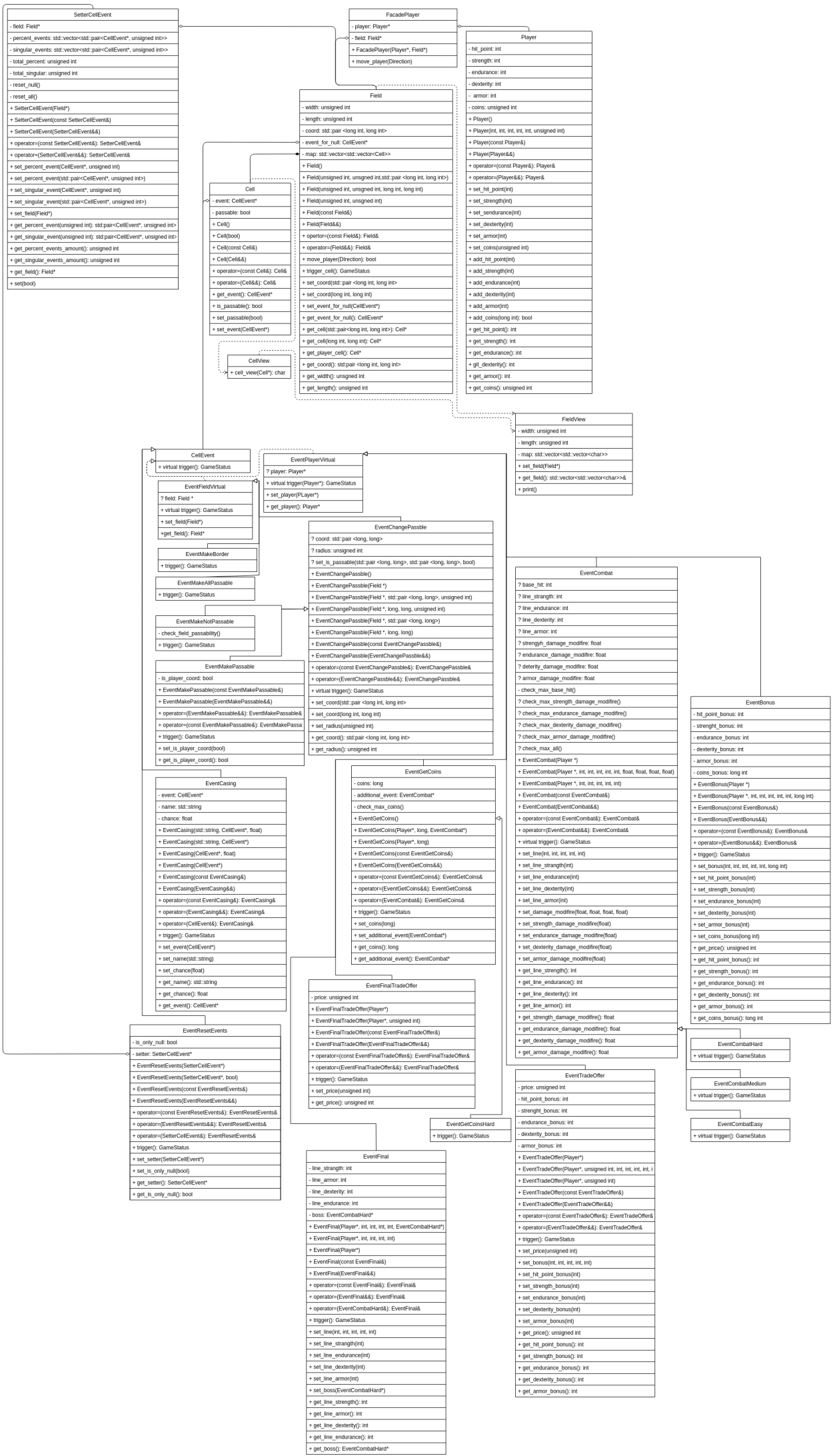
- Координаты игрока, метод движения игрока move_player(Diraction d) и метод активации события trigger_cell() были перенесены из Player в Field. Из Player была удалена указатель на поле field. В Field добавил указатель event_for_null, по которому хранится CellEvent, который активируется если в клетке, которой стоит игрок, не хранится указатель события.

- Был сделан фасад FacadePlayer. Он хранит указатель на поле и игрока. Среди методов пока только конструктор и move_player(Diraction d), который двигает игрока и активирует событие.
- Был сделан класс SetterCellEvent. Он получает настройки на то, какими событиями и как заполнять поле, и заполняет это поле при активации метода set(bool is_only_null).

Выводы.

Были изучены интерфейс и динамический полиморфизм и их работа на языке C++.

ПРИЛОЖЕНИЕ А
UML-ДИАГРАММА МЕЖКЛАССОВЫХ ОТНОШЕНИЙ


SetterCellEvent
- field: Field*
- percent_events: std::vector<std::pair<CellEvent*, unsigned int>>
- singular_events: std::vector<std::pair<CellEvent*, unsigned int>>
- total_percent: unsigned int
- total_singular: unsigned int
- reset_null()
- reset_all()
+ SetterCellEvent(Field*)
+ SetterCellEvent(const SetterCellEvent&)
+ SetterCellEvent(SetterCellEvent&&)
+ operator=(const SetterCellEvent&): SetterCellEvent&
+ operator=(SetterCellEvent&&): SetterCellEvent&
+ set_percent_event(CellEvent*, unsigned int)
+ set_percent_event(std::pair<CellEvent*, unsigned int>)
+ set_singular_event(CellEvent*, unsigned int)
+ set_singular_event(std::pair<CellEvent*, unsigned int>)
+ set_field(Field*)
+ get_percent_event(unsigned int): std:pair<CellEvent*, unsigned int>
+ get_singular_event(unsigned int): std:pair<CellEvent*, unsigned int>
+ get_percent_events_amount(): unsigned int
+ get_singular_events_amount(): unsigned int
+ get_field(): Field*
+ set(bool)
FacadePlayer
- player: Player*
- field: Field*
+ FacadePlayer(Player*, Field*)
+ move_player(Direction)
Player
- hit_point: int
- strength: int
- endurance: int
- dexterity: int
- armor: int
- coins: unsigned int
+ Player()
+ Player(int, int, int, int, int, unsigned int)
+ Player(const Player&)
+ Player(Player&&)
+ operator=(const Player&): Player&
+ operator=(Player&&): Player&
+ set_hit_point(int)
+ set_strength(int)
+ set_sendurance(int)
+ set_dexterity(int)
+ set_armor(int)
+ set_coins(unsigned int)
+ add_hit_point(int)
+ add_strength(int)
+ add_endurance(int)
+ add_dexterity(int)
+ add_armor(int)
+ add_coins(long int): bool
+ get_hit_point(): int
+ get_strength(): int
+ get_endurance(): int
+ git_dexterity(): int
+ get_armor(): int
+ get_coins(): unsigned int
Field
- width: unsigned int
- length: unsigned int
- coord: std::pair <long int, long int>
- event_for_null: CellEvent*
- map: std::vector<std::vector<Cell>>
+ Field()
+ Field(unsigned int, unsigned int,std::pair <long int, long int>)
+ Field(unsigned int, unsigned int, long int, long int)
+ Field(unsigned int, unsigned int)
+ Field(const Field&)
+ Field(Field&&)
+ opertor=(const Field&): Field&
+ operator=(Field&&): Field&
+ move_player(Direction): bool
+ trigger_cell(): GameStatus
+ set_coord(std::pair <long int, long int>
+ set_coord(long int, long int)
+ set_event_for_null(CellEvent*)
+ get_event_for_null(): CellEvent*
+ get_cell(std::pair<long int, long int>): Cell*
+ get_cell(long int, long int): Cell*
+ get_player_cell(): Cell*
+ get_coord(): std:pair <long int, long int>
+ get_width(): unsgned int
+ get_length(): unsigned int
Cell
- event: CellEvent*
- passable: bool
+ Cell()
+ Cell(bool)
+ Cell(const Cell&)
+ Cell(Cell&&)
+ operator=(const Cell&): Cell&
+ operator=(Cell&&): Cell&
+ get_event(): CellEvent*
+ is_passable(): bool
+ set_passable(bool)
+ set_event(CellEvent*)
CellView
+ cell_view(Cell*): char
FieldView
- width: unsigned int
- length: unsigned int
- map: std::vector<std::vector<char>>
+ set_field(Field*)
+ get_field(): std::vector<std::vector<char>>&
+ print()
CellEvent
+ virtual trigger(): GameStatus
EventPlayerVirtual
? player: Player*
+ virtual trigger(Player*): GameStatus
+ set_player(PLayer*)
+ get_player(): Player*
EventFieldVirtual
? field: Field *
+ virtual trigger(): GameStatus
+ set_field(Field*)
+get_field(): Field*
EventMakeBorder
+ trigger(): GameStatus
EventMakeAllPassable
+ trigger(): GameStatus
EventMakeNotPassable
- check_field_passability()
+ trigger(): GameStatus
EventMakePassable
- is_player_coord: bool
+ EventMakePassable(const EventMakePassable&)
+ EventMakePassable(EventMakePassable&&)
+ operator=(EventMakePassable&&): EventMakePassable&
+ operator=(const EventMakePassable&): EventMakePassa
+ trigger(): GameStatus
+ set_is_player_coord(bool)
+ get_is_player_coord(): bool
EventChangePassble
? coord: std::pair <long, long>
? radius: unsigned int
? set_is_passable(std::pair <long, long>, std::pair <long, long>, bool)
+ EventChangePassble()
+ EventChangePassble(Field *)
+ EventChangePassble(Field *, std::pair <long, long>, unsigned int)
+ EventChangePassble(Field *, long, long, unsigned int)
+ EventChangePassble(Field *, std::pair <long, long>)
+ EventChangePassble(Field *, long, long)
+ EventChangePassble(const EventChangePassble&)
+ EventChangePassble(EventChangePassble&&)
+ operator=(const EventChangePassble&): EventChangePassble&
+ operator=(EventChangePassble&&): EventChangePassble&
+ virtual trigger(): GameStatus
+ set_coord(std::pair <long int, long int>
+ set_coord(long int, long int)
+ set_radius(unsigned int)
+ get_coord(): std:pair <long int, long int>
+ get_radius(): unsigned int
EventCombat
? base_hit: int
? line_strangth: int
? line_endurance: int
? line_dexterity: int
? line_armor: int
? strengyh_damage_modifire: float
? endurance_damage_modifire: float
? deterity_damage_modifire: float
? armor_damage_modifire: float
- check_max_base_hit()
? check_max_strength_damage_modifire()
? check_max_endurance_damage_modifire()
? check_max_dexterity_damage_modifire()
? check_max_armor_damage_modifire()
? check_max_all()
+ EventCombat(Player *)
+ EventCombat(Player *, int, int, int, int, int, float, float, float, float)
+ EventCombat(Player *, int, int, int, int, int)
+ EventCombat(const EventCombat&)
+ EventCombat(EventCombat&&)
+ operator=(const EventCombat&): EventCombat&
+ operator=(EventCombat&&): EventCombat&
+ virtual trigger(): GameStatus
+ set_line(int, int, int, int, int)
+ set_line_strength(int)
+ set_line_endurance(int)
+ set_line_dexterity(int)
+ set_line_armor(int)
+ set_damage_modifire(float, float, float, float)
+ set_strength_damage_modifire(float)
+ set_endurance_damage_modifire(float)
+ set_dexterity_damage_modifire(float)
+ set_armor_damage_modifire(float)
+ get_line_strength(): int
+ get_line_endurance(): int
+ get_line_dexterity(): int
+ get_line_armor(): int
+ get_strength_damage_modifire(): float
+ get_endurance_damage_modifire(): float
+ get_dexterity_damage_modifire(): float
+ get_armor_damage_modifire(): float
EventBonus
- hit_point_bonus: int
- strenght_bonus: int
- endurance_bonus: int
- dexterity_bonus: int
- armor_bonus: int
- coins_bonus: long int
+ EventBonus(Player *)
+ EventBonus(Player *, int, int, int, int, int, long int)
+ EventBonus(const EventBonus&)
+ EventBonus(EventBonus&&)
+ operator=(const EventBonus&): EventBonus&
+ operator=(EventBonus&&): EventBonus&
+ trigger(): GameStatus
+ set_bonus(int, int, int, int, int, long int)
+ set_hit_point_bonus(int)
+ set_strength_bonus(int)
+ set_endurance_bonus(int)
+ set_dexterity_bonus(int)
+ set_armor_bonus(int)
+ set_coins_bonus(long int)
+ get_price(): unsigned int
+ get_hit_point_bonus(): int
+ get_strength_bonus(): int
+ get_endurance_bonus(): int
+ get_dexterity_bonus(): int
+ get_armor_bonus(): int
+ get_coins_bonus(): long int
EventCasing
- event: CellEvent*
- name: std::string
- chance: float
+ EventCasing(std::string, CellEvent*, float)
+ EventCasing(std::string, CellEvent*)
+ EventCasing(CellEvent*, float)
+ EventCasing(CellEvent*)
+ EventCasing(const EventCasing&)
+ EventCasing(EventCasing&&)
+ operator=(const EventCasing&): EventCasing&
+ operator=(EventCasing&&): EventCasing&
+ operator=(CellEvent&): EventCasing&
+ trigger(): GameStatus
+ set_event(CellEvent*)
+ set_name(std::string)
+ set_chance(float)
+ get_name(): std::string
+ get_chance(): float
+ get_event(): CellEvent*
EventGetCoins
- coins: long
- additional_event: EventCombat*
- check_max_coins()
+ EventGetCoins()
+ EventGetCoins(Player*, long, EventCombat*)
+ EventGetCoins(Player*, long)
+ EventGetCoins(const EventGetCoins&)
+ EventGetCoins(EventGetCoins&&)
+ operator=(const EventGetCoins&): EventGetCoins&
+ operator=(EventGetCoins&&): EventGetCoins&
+ operator=(EventCombat&): EventGetCoins&
+ trigger(): GameStatus
+ set_coins(long)
+ set_additional_event(EventCombat*)
+ get_coins(): long
+ get_additional_event(): EventCombat*
EventFinalTradeOffer
- price: unsigned int
+ EventFinalTradeOffer(Player*)
+ EventFinalTradeOffer(Player*, unsigned int)
+ EventFinalTradeOffer(const EventFinalTradeOffer&)
+ EventFinalTradeOffer(EventFinalTradeOffer&&)
+ operator=(const EventFinalTradeOffer&): EventFinalTradeOffer&
+ operator=(EventFinalTradeOffer&&): EventFinalTradeOffer&
+ trigger(): GameStatus
+ set_price(unsigned int)
+ get_price(): unsigned int
EventGetCoinsHard
+ trigger(): GameStatus
EventResetEvents
- is_only_null: bool
- setter: SetterCellEvent*
+ EventResetEvents(SetterCellEvent*)
+ EventResetEvents(SetterCellEvent*, bool)
+ EventResetEvents(const EventResetEvents&)
+ EventResetEvents(EventResetEvents&&)
+ operator=(const EventResetEvents&): EventResetEvents&
+ operator=(EventResetEvents&&): EventResetEvents&
+ operator=(SetterCellEvent&): EventResetEvents&
+ trigger(): GameStatus
+ set_setter(SetterCellEvent*)
+ set_is_only_null(bool)
+ get_setter(): SetterCellEvent*
+ get_is_only_null(): bool
EventFinal
- line_strangth: int
- line_armor: int
- line_dexterity: int
- line_endurance: int
- boss: EventCombatHard*
+ EventFinal(Player*, int, int, int, int, EventCombatHard*)
+ EventFinal(Player*, int, int, int, int)
+ EventFinal(Player*)
+ EventFinal(const EventFinal&)
+ EventFinal(EventFinal&&)
+ operator=(const EventFinal&): EventFinal&
+ operator=(EventFinal&&): EventFinal&
+ operator=(EventCombatHard&): EventFinal&
+ trigger(): GameStatus
+ set_line(int, int, int, int, int)
+ set_line_strength(int)
+ set_line_endurance(int)
+ set_line_dexterity(int)
+ set_line_armor(int)
+ set_boss(EventCombatHard*)
+ get_line_strength(): int
+ get_line_armor(): int
+ get_line_dexterity(): int
+ get_line_endurance(): int
+ get_boss(): EventCombatHard*
EventTradeOffer
- price: unsigned int
- hit_point_bonus: int
- strenght_bonus: int
- endurance_bonus: int
- dexterity_bonus: int
- armor_bonus: int
+ EventTradeOffer(Player*)
+ EventTradeOffer(Player*, unsigned int, int, int, int, int, i
+ EventTradeOffer(Player*, unsigned int)
+ EventTradeOffer(const EventTradeOffer&)
+ EventTradeOffer(EventTradeOffer&&)
+ operator=(const EventTradeOffer&): EventTradeOffer&
+ operator=(EventTradeOffer&&): EventTradeOffer&
+ trigger(): GameStatus
+ set_price(unsigned int)
+ set_bonus(int, int, int, int, int)
+ set_hit_point_bonus(int)
+ set_strength_bonus(int)
+ set_endurance_bonus(int)
+ set_dexterity_bonus(int)
+ set_armor_bonus(int)
+ get_price(): unsigned int
+ get_hit_point_bonus(): int
+ get_strength_bonus(): int
+ get_endurance_bonus(): int
+ get_dexterity_bonus(): int
+ get_armor_bonus(): int
EventCombatHard
+ virtual trigger(): GameStatus
EventCombatMedium
+ virtual trigger(): GameStatus
EventCombatEasy
+ virtual trigger(): GameStatus