## **HW2 – Wonderland**

В настольной игре Wonderland несколько игроков по очереди добиваются главной цели – избежать Волшебная страна. Побеждает игрок, который побеждает в Стране чудес.

## **Game board and tiles**

Игровая доска состоит из набора взаимосвязанных плит:

**Board b;**

**auto** t1 = std::make\_shared<Tile>(**"Narrow path"**);

**auto** t2 = std::make\_shared<Tile>(**"Old tree"**);

**auto** t3 = std::make\_shared<Tile>(**"Rabbit hole"**);

**auto** t4 = std::make\_shared<Tile>(**"Bottom of rabbit hole"**);

**auto** t5 = std::make\_shared<Tile>(**"Wonderland woods"**);

**auto** exit = std::make\_shared<ExitWonderlandTile>();

b.append(t1).append(t2).append(t3).append(t4).append(t5).append(exit);

Существует специальная плитка **ExitWonderlandTile**, где пользователи могут вводить, если у них есть определенные типы игральных карт. Для класса **Tile** implement method **canEnter(Player)**, который возвращает **true** если данный игрок может ввести данный **Tile**, в противном случае - **false**.

## **Players, player moves, move behaviors**

Игроки должны знать свое местоположение:

std::unique\_ptr<Player> alice = std::make\_unique<Player>(

"Alice", b.getStartTile());

Чтобы включить движение игроков, создайте методы **forward ()** и **backward ().** У каждого игрока есть **vitality** ценность, которая говорит нам о том, как устал игрок, его значение по умолчанию равно **3**.

Игроки могут иметь разные способы передвижения:

• **Нормальные ходы** - (по умолчанию), жизненная сила уменьшается на **1** при каждом следующем ходу. Как только vitality будет 0, метод **move ()** должен возвращать исключение с сообщением “I am too tired to move”.

• **Unlimited moves**- такой игрок никогда не устает и может бесконечно перемещаться по доске.

Выполните метод **Player.sleep ()**, который восстанавливает значение жизненной силы по умолчанию:

alice->sleep();

alice->getVitality(); // this should return 3

## **Playing Cards**

Игральная карта (abstract base class **Card**)- это элемент, который может быть расположен на какой-либо плитке. Игроки, которые расположенные на одной плитке, могут выбирать игровые карты. Могут быть несколько игральных карт (не обязательно уникальный) на одной плитке. Используйте структуру типа **FIFO** для хранения набора карт на каждой плитке. Методы **tile.addCard (Card c**) и **tile.getFirstCard ()** должны быть реализованы. Способ реализации **tile.getCards ()**, который возвращает коллекцию карт этой плитки, сохраняя их порядок. **Card** должна быть

абстрактного базового класса, так что не должно быть возможности создавать экземпляры этого класса.

std::shared\_ptr<Card> pc = std::make\_shared<PrizeCard>();

t1->addCard(pc);

t1->addCard(pc);

**auto** firstCard = t1->getFirstCard();

**auto** tileCards = t1->getCards();

Игрок может иметь до **5** карт в любой момент времени. Методы реализации:

**Player.pickUp ()** - позволяет игроку выбрать первую карту из текущей плитки. Если игрок уже 5 карт, метод должен выдать исключение.

**Player.getCards ()** - возвращает коллекцию карт игрока.

**Player.getCardsByName (name)** - возвращает диапазон карточек с заданным именем.

**Player.getCardByName (name)** - возвращает карту с указанным именем.

**Player.drop (Card c)** - отбрасывает карту, выпавшая карта становится последней карточкой текущей плитки

alice->pickUp(); *// picks up a prize card*

alice->pickUp(); *// picks up a prize card*

**auto** prizeCards = alice->getCardsByName(**"prize"**);

**auto** prizeCard = alice->getCardByName(**"prize"**);

**auto** aliceCards = alice->getCards();

alice->drop(prizeCard);

alice->drop(prizeCards);

Внедрите методы **onPickedUp ()** и **onDroped ()** таким образом, чтобы для разных типов игровые карты можно переопределить эти методы.

Существуют различные виды игральных карт:

• **ModifierCard** - они влияют на игрока, если игрок несет такую карту.

• **ActionCard** - они оказывают одноразовое воздействие на игрока. Когда такая карта подхвачена, некоторые происходит событие с игроком, и такая карта сразу исчезает.

Добавьте абстрактные базовые классы **ModifierCard** и **ActionCard** в иерархию классов игральных карт.

## **Modifier Cards**

• **PrizeCard** - дает 1 очко игроку.

• **VitalityCard** - пока игрок несет эту карту, увеличивает текущее значение жизненной силы на **1**, а также максимальное значение жизненной силы на **1**. Если игрок сбросит карту жизнеспособности, ее эффект исчезает, и максимальное значение жизненной силы должно быть установлено на **3**.

**std::shared\_ptr<Card> vc = std::make\_shared<VitalityCard>();**

t2->addCard(vc);

alice->forward();

alice->getVitality(); *// returns 2*

alice->getMaxVitality(); *// returns 3*

alice->pickUp();

alice->getVitality(); *// returns 3*

alice->getMaxVitality(); *// returns 4*

alice->drop(alice->getCardByName(**"vitality"**));

alice->getVitality(); *// returns 3*

alice->getMaxVitality(); *// returns 3*

• **StickyCard** - карта, которая не дает никаких преимуществ игроку, но невозможно отбросить такую card - метод **drop ()** должен вызывать исключение.

• **DispelCard** - карта, позволяющая игроку войти в **ExitWonderlandTile** и завершить игра. Когда игрок покинет Страну Чудес, напечатайте количество **PrizeCard**, которое игрок есть.

• **RestlessExplorerCard** - игрок, который выбрал эту карту, никогда не устает. Если игрок сбросил карты, его эффект исчезает.

## **Action Cards**

• **DistractCard** - если игрок выбирает такую карту, любая другая случайно выбранная карта отбрасывается.

• **RewindCard** - игрок, который выбрал эту карту, перемещается в начальную черепицу игры.

• **ZippyCard** - игрок, который выбрал эту карту, увеличивает значение жизненной силы на 1.

• **HypnoticCard** - игрок, который выбрал эту карту, теряет всю свою жизнеспособность.

Создайте тип игроков FlyingPlayer, которые никогда не устают, но могут иметь только до 3 карт.

**88% ActionCard and ModifierCard types were not included in the class hierarchy**