
TP_6_Noté

Guillaume BELLAIZE

Quentin MOALIC

**Déroulement d'un tour de jeu** :

    - Lancer des dés.
    - Avancer le pion sur le plateau.
    - Exécuter l'action en fonction de la case sur laquelle le joueur s'arrête.
- **Types de Cases et leurs Règles :**
    - **Propriétés** (terrain, gare, service public) :
        - Acheter une propriété si elle n'appartient à personne.
        - Payer un loyer si elle appartient à un autre joueur.
        - Construire des maisons ou hôtels lorsqu'un joueur possède tous les terrains d'une même couleur.
    - **Cartes spéciales** :
        - Tirer une carte Chance ou Caisse de Communauté et appliquer ses effets.
    - **Taxes et Impôts** :
        - Case Taxe de luxe ou Impôts sur le revenu.
    - **Prison** :
        - Règles pour aller en prison.
        - Conditions pour sortir de prison (payer une amende, faire un double, utiliser une carte spéciale).
    - **Autres cases** :
        - Case Parc gratuit (neutre).
        - Case Départ (gain de 200 monos).
- **Conditions de Victoire :**
    - Dernier joueur restant sans être en faillite.

**Lancer des dés**

Grâce à la classe Dice, nous pouvons utiliser les dés pour obtenir un résultat de dés aléatoire

```
int number = DDice.roll() + DDice.roll();
```

Grâce à cette ligne, j'obtiens la somme du chiffre des 2 dés.

**Avancer des Player sur le plateau**

```
unsigned int currentPlayerPosition = player->getPosition();
unsigned int newPosition = (currentPlayerPosition + number) % TAILLE_PLATEAU;
player->setPosition(newPosition);
```

Avec le number (somme des chiffres du dés) nous allons actualiser la valeur de la position en prenant soin de ne pas avoir de dépassement grâce au modulo.

**Execution de l'action en fonction de la position**

Après avoir actualiser la position, nous récupérons la case correspondante :

```
Case* CaseCourante = &plateau[newPosition];
```

Puis a l'aide d'un de la fonction « switch case », je regarde le type de la case pour faire l'action correspondante (Voir le programme sur github dans le fichier Game.cpp -> Game::play() ).

**Type de cases et leurs règles : Propriété**

**Comment acheter une propriété**

Voici le code correspondant à cette fonction :

```
if (CaseCourante->getAcheteur() == nullptr && CaseCourante-
>isAchetable())
{
    std::cout << "You have : " << (int) player->getMoneyValue()
<< std::endl;
    if (player->getMoneyValue() < CaseCourante->getPrix())
    {
        std::cout << "Sorry you cannot buy it yet" << std::endl;
    }
    else {
        std::cout << "This area has not been sold yet, do you
want to buy it ? y or n" << std::endl;
        std::string response;
        std::getline(std::cin, response);

        // Check the user's response
        if (response == "y" || response == "Y") {
            std::cout << "You chose to but it, CONGRATULATION!"
<< std::endl;

            CaseCourante->setAcheteur(player);
            CaseCourante->setAchetable(0);
            CaseCourante->setVendu(1);

            std::cout << "You now have : " << player-
>setDeductMoney(CaseCourante->getPrix()) << " mono" <<
std::endl;
        }
        else if (response == "n" || response == "N") {
            std::cout << "You chose not to continue." <<
std::endl;
```

```
            // Add the action you want to perform for "no" here
        }
        else {
            throw std::runtime_error("An error occurred!");
        }
    }

}
```

**Payer un loyer**

Voici le code correspondant à cette fonction :

```
else if (CaseCourante->getAcheteur() != player && CaseCourante-
>isVendu()) {
    //Else : The player is gonna pay to stay in this property
    std::cout << "You are at the " << CaseCourante-
>getCaseName() << "The cost rent is : " << CaseCourante-
>getPrix() << " mono" << std::endl;
    std::cout << "You have : " << player-
>setDeductMoney(CaseCourante->getTaxe()) << " mono" <<
std::endl;
    Player* proprietaire = CaseCourante->getAcheteur();
    proprietaire->addMoney((int) CaseCourante->getPrix());
}
```

**Construire des maisons ou hôtels lorsqu'un joueur possède tous les terrains d'une même couleur.**

Grâce à des structures je sauvegarde le nombre de terrain par couleur acheter par le joueur.

```
struct ColorHouseCount {
    Couleur color;                  ///< Color group.
    unsigned int terrainCount;      ///< Number of properties in the group.
    bool hasHotel;                  ///< Indicates if any property has a hotel.
    unsigned int maxNumberOfTerrain; ///< Maximum number of properties in the group.
    unsigned int houseCount;        ///< Total number of houses in the group.
};
```

Elle me permet aussi de compter le nombre de maison construite sur un groupe de couleur ainsi que la présence d'un immeuble.

L'algorithme suivant nous permet d'acheter des maisons si on a tous les terrains d'un groupe de couleur, au bout de 4 maisons sur une couleur on construit un immeuble, il sert aussi à dire à l'utilisateur quel groupe il a réussi à compléter.

```
else if (CaseCourante->getAcheteur() == player)
{
    std::cout << "You are at home." << std::endl;
```

```
    uint8_t countRouge = 0;
    uint8_t countBleu = 0;
    uint8_t countCian = 0;
    uint8_t countVert = 0;
    uint8_t countMauve = 0;
    uint8_t countRose = 0;
    uint8_t countOrange = 0;
    uint8_t countJaune = 0;
    bool thereIsAtLeastOne = 0;
    // Counting how many terrain the player have by color
    for (int i = 0; i < TAILLE_PLATEAU; i++) {
        Couleur c = plateau[i].getCouleur();
        if (plateau[i].getAcheteur() == player && c != NO_COLOR)
{
            switch (c) {
            case ROUGE:
                ++countRouge; // Increment Red
                break;
            case BLEU:
                ++countBleu; // Increment Blue
                break;
            case CIAN:
                ++countCian; // Increment Cyan
                break;
            case VERT:
                ++countVert; // Increment Green
                break;
            case MAUVE:
                ++countMauve; // Increment Mauve
                break;
            case ROSE:
                ++countRose; // Increment Pink
                break;
            case ORANGE:
                ++countOrange; // Increment Orange
                break;
            case JAUNE:
                ++countJaune; // Increment Yellow
                break;
            case NO_COLOR:
                // Do nothing for NO_COLOR
                break;
            default:
                std::cerr << "Unknown Color!" << std::endl;
                break;
            }
        }
    }
    // Say to the user, that he has completed at least one group
of color if there is some
```

```
    if (countRouge >= 3) {
        std::cout << "You have all the Rouge color case!" <<
std::endl;
        thereIsAtLeastOne = true;
    }
    if (countOrange >= 3) {
        std::cout << "You have all the Orange color case!" <<
std::endl;
        thereIsAtLeastOne = true;
    }
    if (countRose >= 3) {
        std::cout << "You have all the Rose color case!" <<
std::endl;
        thereIsAtLeastOne = true;
    }
    if (countCian >= 3) {
        std::cout << "You have all the Cian color case!" <<
std::endl;
        thereIsAtLeastOne = true;
    }
    if (countBleu >= 2) {
        std::cout << "You have all the Bleu color case!" <<
std::endl;
        thereIsAtLeastOne = true;
    }
    if (countVert >= 3) {
        std::cout << "You have all the Vert color case!" <<
std::endl;
        thereIsAtLeastOne = true;
    }
    if (countMauve >= 2) {
        std::cout << "You have all the Mauve color case!" <<
std::endl;
        thereIsAtLeastOne = true;
    }
    if (countJaune >= 3) {
        std::cout << "You have all the Jaune color case!" <<
std::endl;
        thereIsAtLeastOne = true;
    }

    // If the player have at list all the cases of a color group
    if (thereIsAtLeastOne) {
        std::cout << "At least one color group is complete!" <<
std::endl;
        std::cout << "Do you want to add a House/Motel ? y or n"
<< std::endl;
        std::string response;

        std::getline(std::cin, response);
```

```
        // Check the user's response
        if (response == "y" || response == "Y") {
            std::cout << "You chose to add a House/Motel!" <<
std::endl;

            // Pick up the color group of the current case and
verify that the player own all the terrain of this color
            Couleur couleurDuGroupeDeMaison = CaseCourante-
>getCouleur();
            for (auto& group : allGroups) {
                if (group.color == couleurDuGroupeDeMaison &&
group.terrainCount >= group.maxNumberOfTerrain) {
                    if (group.houseCount < 4) {
                        CaseCourante->addHouse();
                        group.houseCount++;
                    }
                    else {
                        CaseCourante->addMotel();
                        group.hasHotel = true;
                    }
                }
            }
        }
        else if (response == "n" || response == "N") {
            std::cout << "You chose not to continue." <<
std::endl;
            // Add the action you want to perform for "no" here
        }
        else {
            throw std::runtime_error("An error occurred! Invalid
input.");
        }
    }
    else {
        std::cout << "No complete color group yet." <<
std::endl;
    }
```

**Cartes spéciales**

Une énumération sert à lister toute les cartes :

```
enum Cards {
    CARTE_CHANCE = 0x00,                    ///< Chance card.
    CARTE_CAISSE_DE_COMMUNAUTE = 0x01,      ///< Community Chest card.
    CARTE_ALLER_EN_PRISON = 0x02,           ///< Go to Jail card.
    CARTE_VOUS_ETES_LIBERE_DE_PRISON = 0x03 ///< Get Out of Jail Free card.
};
```

Tirer une carte Chance ou Caisse de Communauté

```
case ALLER_EN_PRISON:
{
    // Code for ALLER_EN_PRISON
    player->addCard(CARTE_ALLER_EN_PRISON);
    player->setPosition(10);
    break;
}
```

```
case CAISSE_DE_COMMU:
{
    // Code for CAISSE_DE_COMMU
    player->addCard(CARTE_CAISSE_DE_COMMUNAUTE);
    break;
}
```

Les cartes sont ensuite stocké dans le joueur via cet attribut :

```
std::vector<Cards> m_cartes;       ///< The cards in the player's possession.
```

**Taxes et Impôts : Case Taxe de luxe ou Impôts sur le revenu.**

L'argent de la taxe et des impôts revient à la banque.

```
case TAXE_DE_LUXE:
{
    // Code for TAXE_DE_LUXE
    std::cout << "You have to pay 10000 mono to the bank!" << std::endl;
    player->setDeductMoney(10000);
    BBank.addMoney(10000);
    break;
}
```

```
case IMPOT_SUR_LE_REVENU:
{
    // Code for IMPOT_SUR_LE_REVENU
    std::cout << "You have to pay 200 mono to the bank!" << std::endl;
    player->setDeductMoney(200);
    BBank.addMoney(200);
    break;
}
```

**Prison**

Règles pour aller en prison :

```
case ALLER_EN_PRISON:
{
    // Code for ALLER_EN_PRISON
    player->addCard(CARTE_ALLER_EN_PRISON);
    player->setPosition(10);
    break;
}
```

Conditions pour sortir de prison (payer une amende, faire un double, utiliser une carte spéciale) :

```
if (player->getInPrison())
{
    std::cout << "Your are in prison, you have 3 option to go out : " << std::endl;
    std::cout << "Option 1 : Pay 50 mono " << std::endl;
    std::cout << "Option 2 : try to exchange a Get out of Jail free card with another player or by using your own" << std::endl;
    std::cout << "Option 3 : Roll the dice and try to catch a double " << std::endl;
```

```
    std::cout << "Now answer by typing 1 or 2 or 3" <<
std::endl;
    std::string response;
    std::getline(std::cin, response);

    // Check the user's response
    if (response == "1") {
        std::cout << "You chose to but it, CONGRATULATION!" <<
std::endl;
        std::cout << "You now have : " << player-
>setDeductMoney(50) << " mono" << std::endl;
        BBank.addMoney(50);
        player->setInPrison(0);
    }
    else if (response == "2") {
        std::cout << "You choose card." << std::endl;
    }
    else if (response == "3") {
        std::cout << "You choose to not pay. Let s see if you
are lucky!" << std::endl;
        if (DDice.roll() == DDice.roll()) {
            std::cout << "You are lucky!" << std::endl;
            player->setInPrison(0);
        }
        else {
            std::cout << "Looser..." << std::endl;
        }
    }
    else {
        throw std::runtime_error("An error occurred!");
    }

}
```

**Autres cases**

Case Parc gratuit (neutre).

La case parc gratuit permet de récupérer l'argent de la banque :

```
case PARC_GRATUIT:
{
    // Code for PARC_GRATUIT
    unsigned int moneyInBank = BBank.getMoney();
    player->addMoney(moneyInBank);
    BBank.deductMoney(moneyInBank);
    break;
}
```

Case Départ (gain de 200 monos).

```
case DEPART:
{
    // Code for DEPART
    std::cout << "CONGRATULATION you won 20000 mono!" << std::endl;
    player->addMoney(20000);
    break;
}
```

**Conditions de Victoire**

Dernier joueur restant sans être en faillite gagne la partie. Le joueur sera mit hors du jeu avec cette méthode :

```
void Game::checkPlayer() {
    for (int i = 0; i < this->m_player_list.size(); i++)
    {
        Player* player = this->m_player_list[i];
        if ((int)player->getMoneyValue() < 0 && !player->getOut()) {
            std::cout << "Player " << player->getName() << " out" << std::endl;
            player->setOut(1);
        }
    }
}
```

Puis quand il ne restera qu'un joueur la partie s'arrête :

```
bool Game::hasEnded()
{
    unsigned int nbPlayer = this->m_player_list.size();

    for (int i = 0; i < this->m_player_list.size(); i++)
    {
        Player* player = this->m_player_list[i];
        if (player->getOut()) {
            nbPlayer--;
        }
    }

    if (nbPlayer == 1) return 1;
    if (nbPlayer > 1 ) return 0;
}
```