جـامعة الحسن الثـاني بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

UNIVERSITE HASSAN II DE CASABLANCA

ECOLE NATIONALE SUPERIEURE DE
L'ENSEIGNEMENT TECHNIQUE DE MOHAMMEDIA

ENSET

Module : Structures de données

Rapport du Project: Tetriste

Réalisé par : M. EL-HADRY Abderrazzak

Encadré par: Pr. Qbadou Mohamed

Année Universitaire : 2023-2024

I. Contexte du projet :

1. Présentation du Jeu :

a. Introduction

Tetriste est un jeu de puzzle simple développé dans le cadre du cours Structures de données. Inspiré du célèbre jeu Tetris, Tetriste propose une expérience de jeu unique où les joueurs doivent aligner des pièces de différentes formes et couleurs pour marquer des points.

b. Concept

Dans Tetriste, le joueur est confronté à un plateau de jeu sous forme d'une ligne droite extensible par la gauche et la droite. Des pièces de différentes et couleurs sont générées aléatoirement et doivent être insérées sur le plateau par le joueur.

c. Objectif

L'objectif principal jeu est de créer des motifs de formes ou de couleurs répétés. Lorsque trois pièces consécutives ont la même couleur ou forme, elles disparaissent du plateau, permettant au joueur de marquer des points. Le jeu se termine lorsque le plateau est plein et qu'il n'est plus possible d'insérer de nouvelles pièces.

d. Mécaniques de jeu

Tetriste propose des mécaniques de jeu pour rendre l'expérience plus intéressante :

- **Insertions des pièces :** Le joueur doit choisir où insérer (Gauche, Droite) chaque nouvelle pièce pour créer des alignements bénéfiques.
- **Décalages :** En plus des insertions, le joueur peut effectuer des décalages à gauche ou droite pour permettre des suppressions de pièces qui ne seraient pas atteignables autrement.

2. Environnement de travail :

a. Environnement matériel :

J'ai utilisé mon ordinateur personnel avec la configuration suivante :

- OS: Windows 10 64 bit
- CPU: i7-5500U
- GPU: GTX 950M 4go
- RAM: 8 go ddr4

b. Environnement logiciel:

Visual Studio Code est un éditeur de code libre et open-source développé par Microsoft.

Diagrams est une application open source en ligne pour la création des diagrammes.

c. Langages de programmation et modélisation :

C++ est un langage de programmation compilé permettant la programmation sous de multiples paradigmes (procédurale, orientée objet, générique).

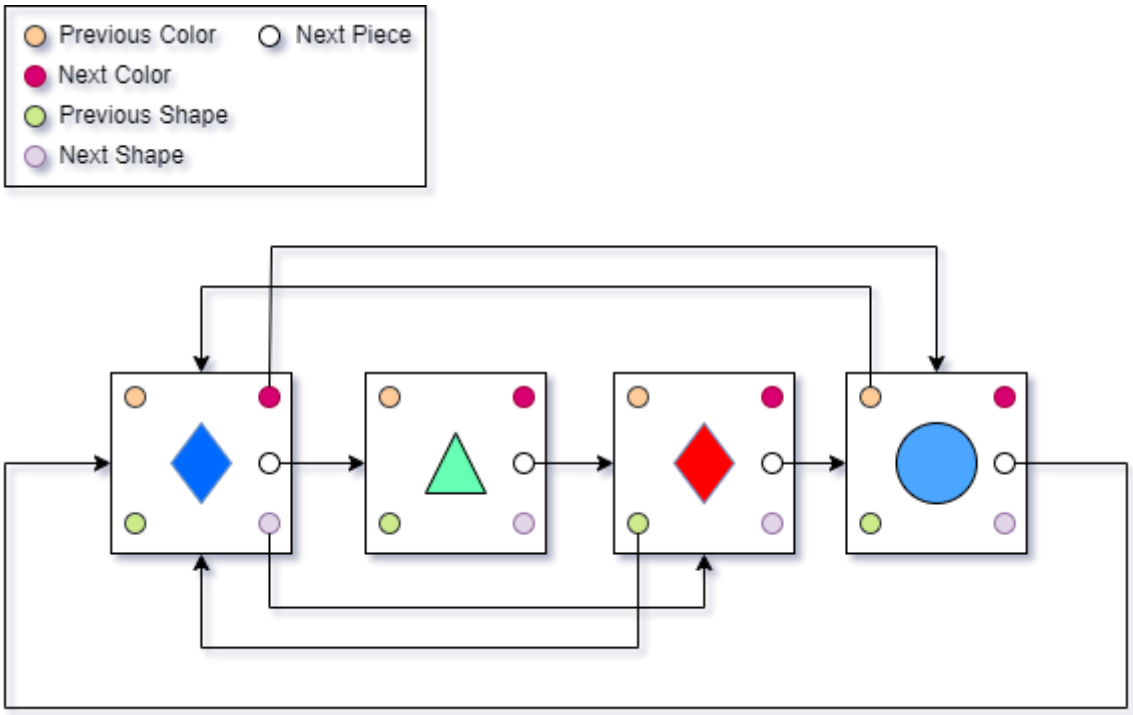
- **bibliothèque** utilisée :

SFML est une interface de programmation bas niveau destinée à construire des jeux vidéo ou des programmes interactifs.

Le langage UML "Unified Modeling Language" est constitué de diagrammes intègres utilisés pour la représentation visuelle des objets, des états et des processus dans un logiciel ou système.

II. Conception et Modélisation :

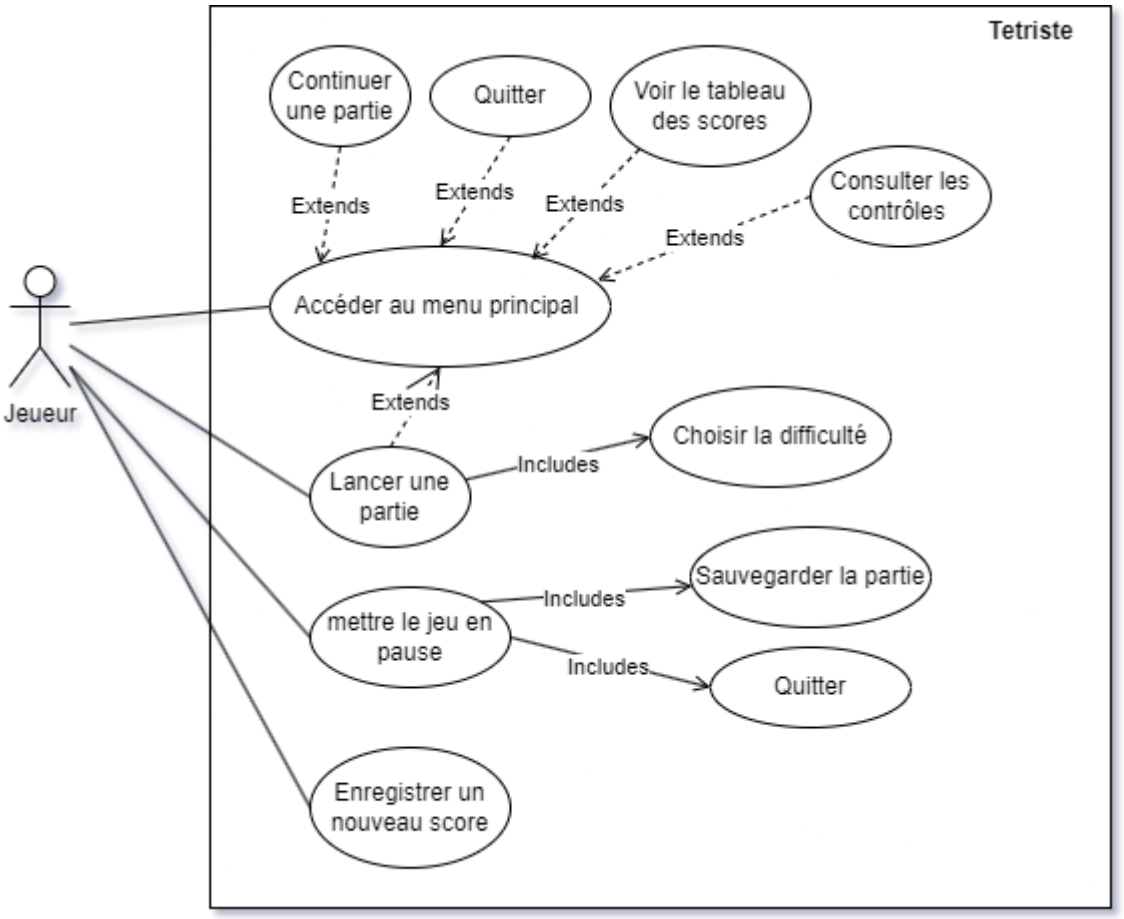
1. Chainage des Pieces :



Le chainage des pièces entre elles dans le jeu Tetriste est réalisé à l'aide de listes chaînées circulaires. Chaque pièce du jeu est liée à la suivante dans la séquence d'insertion, formant ainsi une liste principale. De plus, chaque pièce est également liée à d'autres pièces ayant la même forme ou couleur grâce à des pointeurs spécifiques, permettant ainsi de naviguer facilement entre les pièces partageant les mêmes caractéristiques. Cette structure de données facilite l'insertion, la suppression et la gestion des pièces dans le jeu.

2. Diagramme de cas d'utilisation :

Dans cette partie, j'ai présenté le diagramme de cas d'utilisation global qui modélise besoins fonctionnels de projet :



III. Implémentation et réalisation

1. Démonstration des interfaces :

a. Menu principal :

L'interface du menu principal du jeu Tetris est simple et intuitive, avec un design minimaliste et des éléments clairement identifiables.

Boutons :

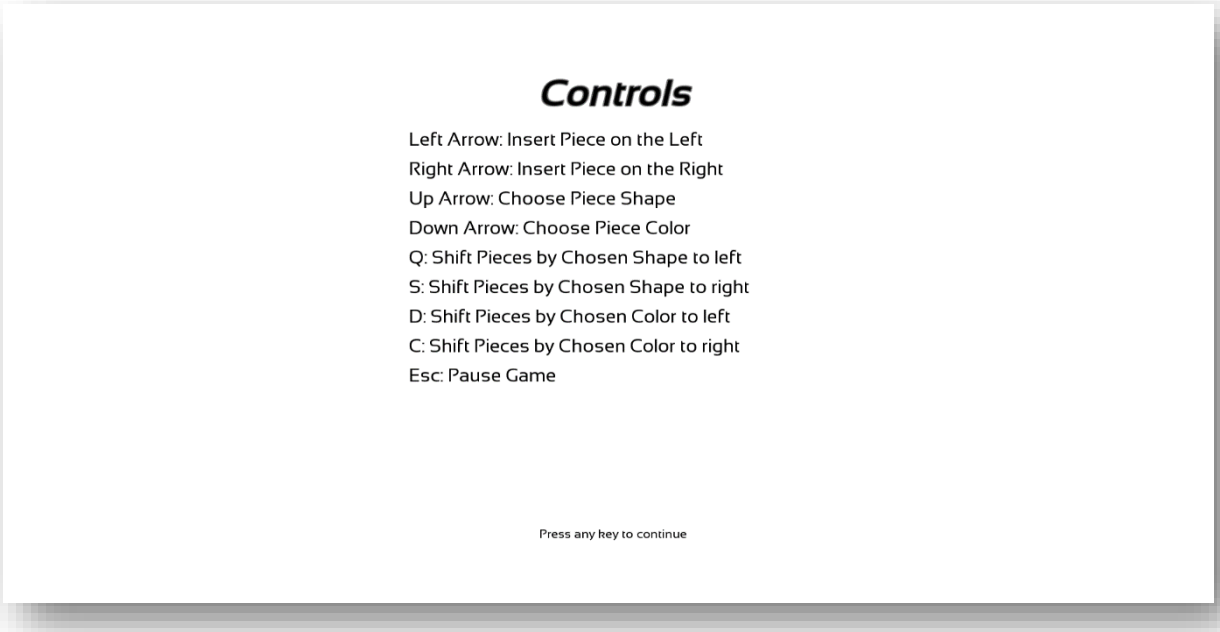
- **Nouveau jeu (New Game) :** Démarre une nouvelle partie de Tetris.
- **Classement (LeaderBoard)** : Affiche le classement des meilleurs scores.
- **Commandes (Controls)** : Explique les commandes du jeu.
- **Quitter (Quit) :** Ferme le jeu.

En cas d'existence d'une partie sauvegardée, on ajoute un nouveau bouton (**Resume Game**) pour continuer la partie.

b. Classement :

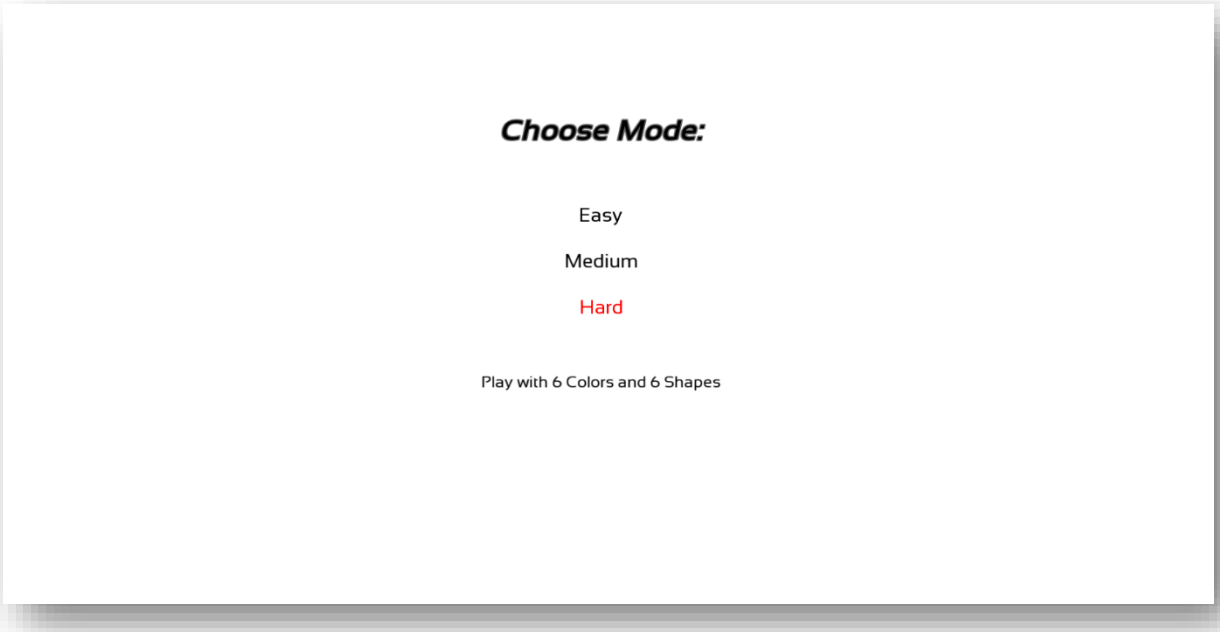
Cette interface permet aux joueurs de voir leur position dans le classement par rapport aux autres joueurs. Le tableau affiche les noms d'utilisateur et leurs scores correspondants, permettant aux joueurs de se comparer et de suivre leur progression.

c. Commandes :



Cette interface joue un rôle essentiel en informant les joueurs sur la façon de contrôler le jeu.

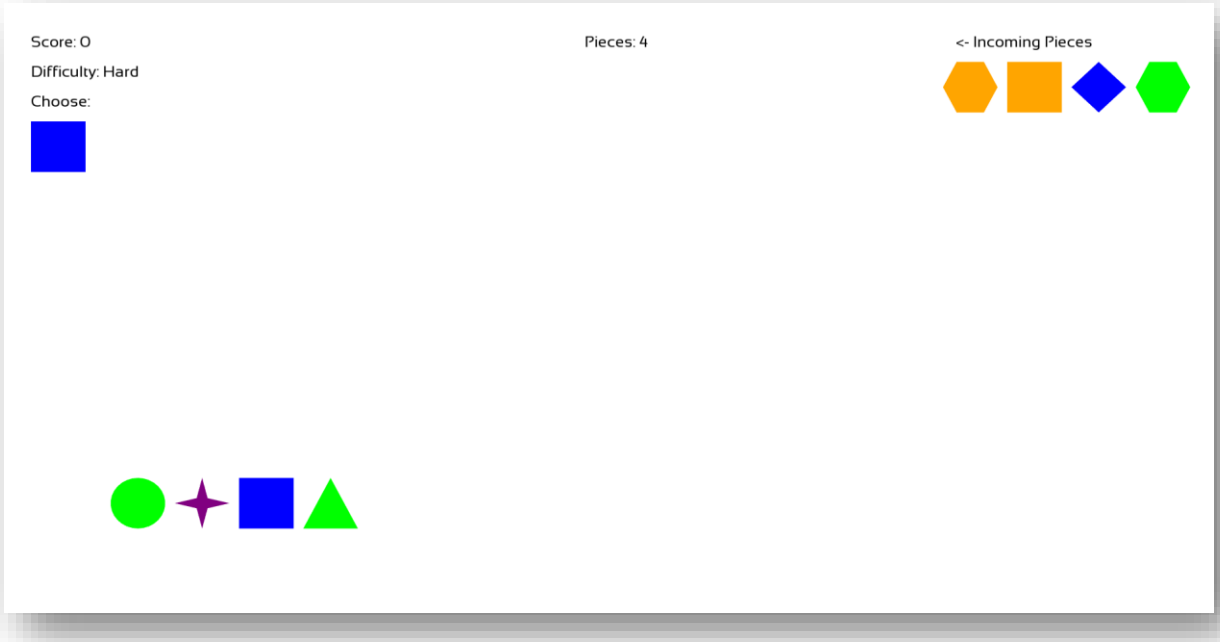
d. Menu des difficultés :



Ce menu permet aux joueurs de personnaliser leur expérience de jeu en choisissant un niveau de difficulté et potentiellement d'autres paramètres. Cela permet aux joueurs de découvrir le jeu progressivement et d'augmenter la difficulté à mesure qu'ils s'améliorent.

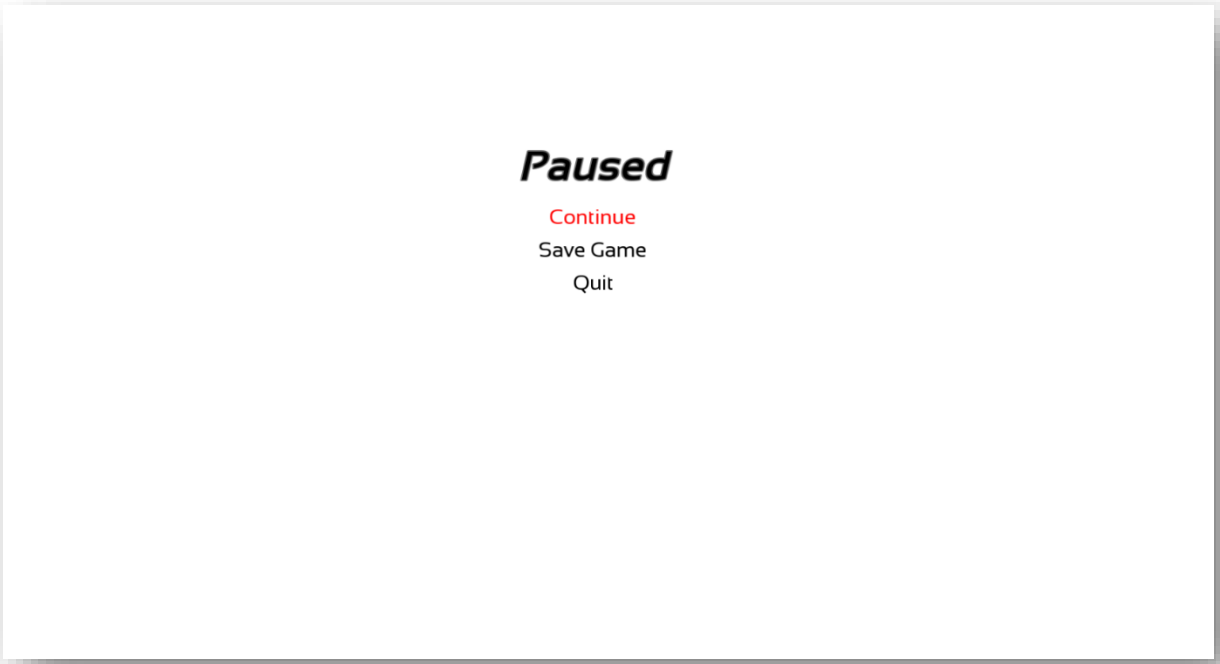
- Facile : Jouer avec 4 couleurs et 4 formes.
- Moyen : Jouer avec 5 couleurs et 5 formes.
- Difficile : Jouer avec 6 couleurs et 6 formes.

e. Partie du jeu :



- **Informations sur le jeu :**
  - **Score :** Affiche le score actuel du joueur.
  - **Pièces :** Indique le nombre de pièces restantes sur le plateau de jeu.
  - **Difficulté :** Affiche le niveau de difficulté sélectionné par le joueur.
  - **Pièces suivantes :** Cette section présente les quatre prochaines formes qui apparaîtront sur le plateau de jeu.
  - **Choisir :** Cette section affiche la forme et la couleur actuellement sélectionnées par le joueur, qui seront insérées sur le plateau de jeu. Le joueur peut utiliser les touches fléchées pour changer sa sélection.

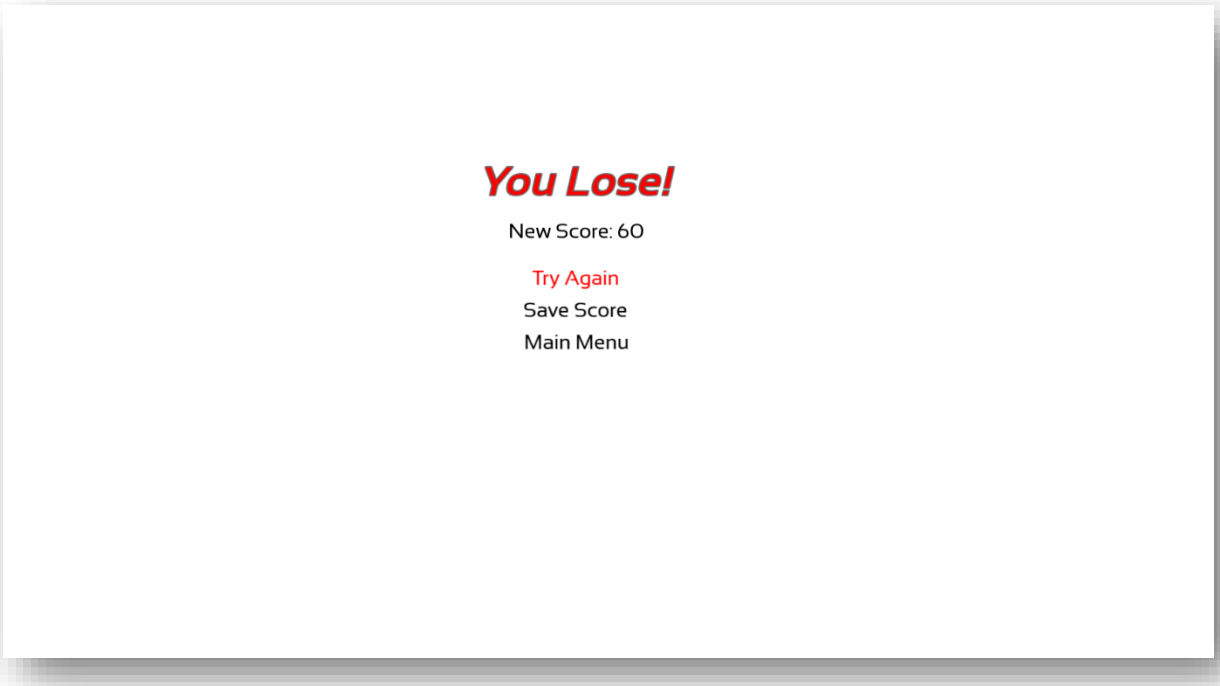
f. Mettre en pause :



Cet écran permet au joueur de mettre le jeu en pause, de le continuer, de le sauvegarder et de le quitter.

- **Paused** : Ce texte indique que le jeu est en pause.
- **Continue** : Ce bouton permet de reprendre le jeu à l'endroit où il a été mis en pause.
- **Save Game** : Ce bouton permet de sauvegarder la partie en cours.
- **Quit** : Ce bouton permet de quitter le jeu et de revenir au menu principal.

g. Fin de la partie :



L'interface de fin de partie propose également plusieurs options au joueur :

- **Réessayer (Try Again) :** Ce bouton permet de redémarrer le niveau ou le jeu pour que le joueur puisse réessayer de le gagner.
- **Sauvegarder le score (Save Score) :** Ce bouton pourrait enregistrer le score de du joueur dans un classement ou le conserver localement sur l'appareil.
- **Menu principal (Main Menu) :** Ce bouton ramènerait le joueur au menu principal du jeu.

2. Test du code sous Valgrind :

Voici les fonctionnalités testées :

```
int main(){
    // test all functionalities
    GameBoard game;
    game.printBoard();
    game.checkMatches();
    cout << "Score: " << game.getScore() << endl;
    cout << "Size: " << game.getSize() << endl;
    game.saveGame();

    // insert 3 pieces
    game.insert(true, new Piece(PColor::BLUE, PShape::SQUARE));
    game.insert(true, new Piece(PColor::BLUE, PShape::TRIANGLE));
    game.insert(true, new Piece(PColor::BLUE, PShape::CIRCLE));
    game.checkMatches();
    // test shifting
    game.shiftByColor(true, PColor::BLUE);
    game.printBoard();
    game.shiftByColor(false, PColor::BLUE);
    game.printBoard();
    game.shiftByShape(true, PShape::SQUARE);
    game.printBoard();

    return 0;
}
```

Le code a été testé sous WSL Ubuntu 22, le résultat été comme suite :

```
==1662==
==1662== HEAP SUMMARY:
==1662==     in use at exit: 0 bytes in 0 blocks
==1662==   total heap usage: 14 allocs, 14 frees, 75,752 bytes allocated
==1662==
==1662== All heap blocks were freed -- no leaks are possible
==1662==
==1662== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
abdex@DESKTOP-5QUF4GB:~$
```

La sortie du test signifie que la gestion de la mémoire du programme semble bonne car aucune fuite de mémoire ou erreur n'a été trouvée.

3. La complexité des fonctions utilisées :

Voici un tableau récapitulatif des complexités temporelles des méthodes utilisés pour gérer le jeu :

Fonction	Complexité temporelle
insert(bool left, Piece *piece)	O(1)
updatePointers()	O(n)
shiftByColor(bool left, PColor color)	O(n)
shiftByShape(bool left, PShape shape)	O(n)
removePiece(Piece *piece)	O(n)
checkMatches()	O(n)

1. **insert(bool left, Piece *piece)**:
    - Cette méthode insère une pièce dans le plateau de jeu.
    - Si la pièce est insérée, elle est ajoutée soit à la tête ou à la fin de la liste chaînée, ce qui a une complexité temporelle $O(1)$.
2. **updatePointers()**:
    - Cette méthode met à jour les pointeurs pour chaque couleur et chaque forme dans le plateau de jeu.
    - Elle parcourt tous les éléments du plateau de jeu une fois, ce qui donne une complexité temporelle $O(n)$, où n est la taille actuelle du plateau de jeu.
3. **shiftByColor(bool left, PColor color)**:
    - Cette méthode déplace les pièces d'une couleur spécifique dans le plateau de jeu vers la gauche ou vers la droite.
    - La complexité temporelle de cette méthode dépend du nombre de pièces de la couleur spécifiée. Si m est le nombre de pièces de la couleur spécifiée, alors la complexité est $O(m)$.
4. **shiftByShape(bool left, PShape shape)**:
    - Cette méthode déplace les pièces d'une forme spécifique dans le plateau de jeu vers la gauche ou vers la droite.
    - La complexité temporelle de cette méthode dépend du nombre de pièces de la forme spécifiée. Si k est le nombre de pièces de la forme spécifiée, alors la complexité est $O(k)$.
5. **removePiece(Piece *piece)**:
    - Cette méthode supprime une pièce spécifique du plateau de jeu.
    - Dans le pire des cas, elle doit parcourir toute la liste pour trouver la pièce à supprimer, ce qui donne une complexité temporelle $O(n)$, où n est la taille actuelle du plateau de jeu.
6. **checkMatches()**:
    - Cette méthode vérifie s'il y a trois pièces ou plus de la même couleur ou forme consécutives dans le plateau de jeu, les supprime et met à jour le score.
    - Dans le pire des cas, elle doit parcourir toute la liste pour vérifier chaque pièce, ce qui donne une complexité temporelle $O(n)$, où n est la taille actuelle du plateau de jeu.

Conclusion

La création et le développement du jeu Tetriste ont été une expérience enrichissante, mettant en avant différentes compétences en programmation et en conception de jeux.

À travers ce projet, j'ai pu explorer les tenants et aboutissants de la création d'un jeu, depuis la conceptualisation jusqu'à la réalisation concrète. La mise en œuvre de structures de données telles que les listes chaînées circulaires a été particulièrement instructive, offrant une compréhension approfondie de la gestion des éléments de jeu et de leurs interactions.

De plus, l'optimisation du code et les tests rigoureux ont permis d'assurer une expérience de jeu fluide et sans erreur pour les utilisateurs.

Enfin, ce projet ouvre la voie à de nouvelles opportunités d'apprentissage et de développement dans le domaine de la programmation de jeux, avec la possibilité d'explorer des fonctionnalités avancées et d'élargir encore plus le jeu pour répondre aux attentes et aux besoins des joueurs.

Perspectives :

- Amélioration des algorithmes afin de réduire leur complexité et d'optimiser les performances globales du jeu.
- Intégration d'une intelligence artificielle capable de recommander les meilleurs coups à jouer.