Rapport de Projet C++/SDL
Casse-brique

L'Université du Littoral Côte d'Opale – ULCO

Année universitaire 2024–2025

Projet réalisé par

Mohammed Amine HIBAOUI

Projet encadré par

Frederic OUEDRAOGO

Table des matières

1 Introduction

Ce rapport présente le travail réalisé durant les séances 3, 4 et 5 dans le cadre du projet INFO7. Le but était de développer un mini-jeu de type casse-brique en C++ en utilisant la librairie SDL2 et en respectant les contraintes imposées (pas de classes, pas de passage par référence, pas de vecteur/tableau statique, etc.).

2 Séance 3 : Représentation et chargement du monde

2.1 Objectifs de la séance

Représenter le monde du jeu et le charger à partir d'un fichier.

2.2 Travail réalisé

- Création de la structure `World` contenant :
  - largeur et hauteur ;
  - une grille 1D de blocs de type `Block`.
- **À faire** : `init_world`, `free_world`, `read`, `write`, `display`.
- Lecture du monde à partir d'un fichier `world.dat` via la fonction `init_world_from_file`.
- Types de blocs : `Empty`, `Border`, `Type1`, `Type2`, `Lose`.
- Test d'affichage du monde en console.

2.3 Explication des fonctions clés

- `init_world` : alloue dynamiquement la structure du monde et initialise les données à zéro. Chaque case de la grille est remplie avec le bloc `Empty`.

```
void init_world(World* world, int w, int h) {
    world->width = w;
    world->height = h;
    world->grid = new Block[w * h];
    for (int i = 0; i < w * h; i++) {
        world->grid[i] = Empty;
    }
}
```

- `free_world` : libère la mémoire allouée au tableau `grid` et évite les pointeurs sauvages en mettant `grid = nullptr`.

```
void free_world(World* world) {
    delete[] world->grid;
    world->grid = nullptr;
}
```

- `read` et `write` : permettent l'accès et la modification d'une case donnée en gérant les erreurs de coordonnées.
- `init_world_from_file` : lit les dimensions puis les caractères du fichier `world.dat` pour initialiser la grille avec les bons types de blocs.
- `display` : affiche la grille en mode texte avec des caractères correspondant aux types de blocs (ex : '.' pour `Empty`, '' pour `Border`, etc.).

3 Séance 4 : Affichage SDL et animation de base

3.1 Objectifs de la séance

Créer un affichage graphique de la grille dans une fenêtre SDL et implémenter une boucle événementielle.

3.2 Explication des fonctions importantes

Affichage graphique SDL : la fonction `display_game` convertit chaque bloc en rectangle de couleur selon le type de bloc, en utilisant les fonctions `draw_fill_rectangle` et `set_color`. La raquette est dessinée sur plusieurs cases horizontales, et la balle est soit une image, soit un carré blanc.

Gestion des entrées clavier : la fonction `keyboard_event` lit les événements clavier (Q, R, Espace, flèches) pour interagir avec le jeu.

Changement de statut : le jeu peut être dans 5 statuts (Begin, Play, Pause, Win, GameOver). Chaque pression sur Espace modifie cet état.

4 Séance 5 : Finalisation du jeu et gestion des rebonds

4.1 Explication détaillée des fonctions principales

Déplacement de la balle : la fonction `move_ball` met à jour les coordonnées de la balle en ajoutant son vecteur de direction (`dx`, `dy`). Le bloc visé est récupéré avec `get_block`. S'il est destructible, la balle rebondit verticalement et le bloc est remplacé par `Empty`. S'il s'agit d'un bord, on regarde l'orientation du choc pour inverser `dx` ou `dy`. Si la balle est juste au-dessus de la raquette, elle rebondit en fonction de la position d'impact (centre, gauche, droite).

```
void move_ball(Game* game) {
    int next_x = game->ball_x + game->ball_dx;
    int next_y = game->ball_y + game->ball_dy;
    Block cible = get_block(game, next_x, next_y);

    if (cible == Type1 || cible == Type2) {
        write(game->world, next_x, next_y, Empty);
        game->score++;
        game->ball_dy *= -1;
    }
    else if (cible == Border) {
        // rebond selon axe horizontal/vertical
    }
    game->ball_x += game->ball_dx;
    game->ball_y += game->ball_dy;
}
```

Délimitation des bordures : toute coordonnée en dehors de la grille retourne `Border` grâce à la fonction `get_block`. Cela permet à la balle de rebondir si elle atteint une limite du monde (haut, gauche, droite).

```
Block get_block(Game* game, int x, int y) {
    if (x < 0 || x >= game->world->width || y < 0 || y >= game->world->heig
        return Border;
    return read(game->world, x, y);
}
```

Destruction de blocs : lorsqu'un bloc `Type1` ou `Type2` est touché, il est supprimé de la grille grâce à `write`, et le score est incrémenté. Le rebond s'effectue automatiquement en inversant `dy` (rebond vertical).

```
if (cible == Type1 || cible == Type2) {
    write(game->world, next_x, next_y, Empty);
    game->score++;
    dy *= -1;
}
```

Fin de partie : la fonction `check_game_status` vérifie si la balle touche un bloc `Lose` (défaite) ou si tous les blocs destructibles ont disparu (victoire). Elle adapte le statut du jeu en conséquence.

```
void check_game_status(Game* game) {
    if (read(game->world, game->ball_x, game->ball_y) == Lose)
        game->statut = GameOver;
    // sinon v rifie qu'il ne reste plus de bloc
}
```

5 Conclusion

Le jeu est maintenant jouable avec des rebonds réalistes, une raquette dynamique, un système de score et une détection de fin de partie. Il peut être amélioré par l'ajout de sprites ou de nouveaux types de blocs.

6 Remerciements

Je tiens à remercier monsieur Frederic OUEDRAOGO pour son encadrement et ses conseils tout au long de ce projet. Merci également à mes camarades de classe pour leurs aide et leurs soutiens.

7 Références

— Documentation officielle de SDL2 : `https://www.libsdl.org/`
— Tutoriels C++ : `https://www.learncpp.com/`
— SDL keycode header : `https://github.com/libsdl-org/SDL/blob/SDL2/include/SDL_keycode.h`

8 Les exécutables du projet

Voici quelques captures d'écran illustrant les différents rendus obtenus lors de l'exécution du projet :

FIGURE 1 – Exécution du jeu Casse-brique avec la balle, la raquette et les blocs.

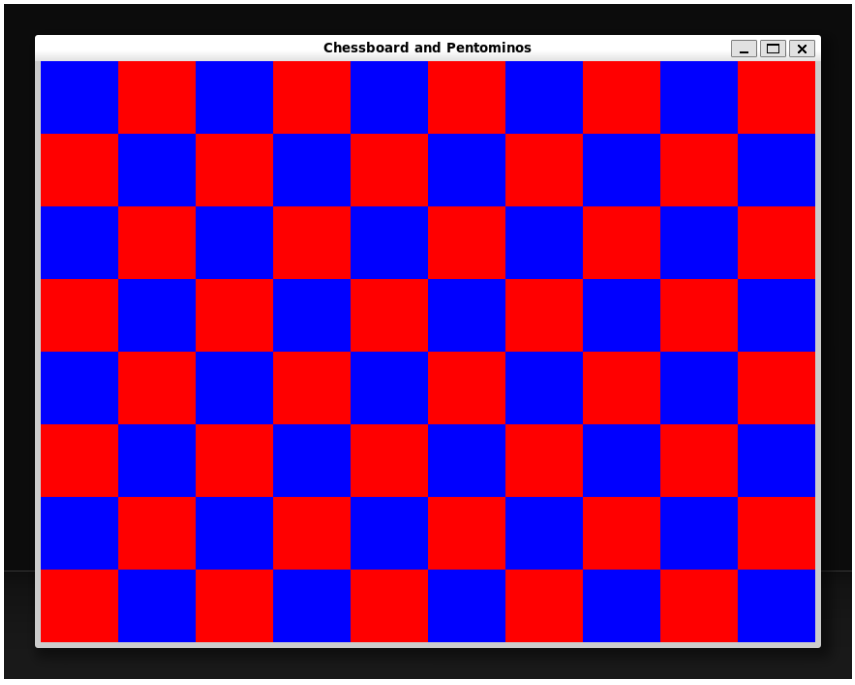

FIGURE 2 – Affichage d'un échiquier en SDL.

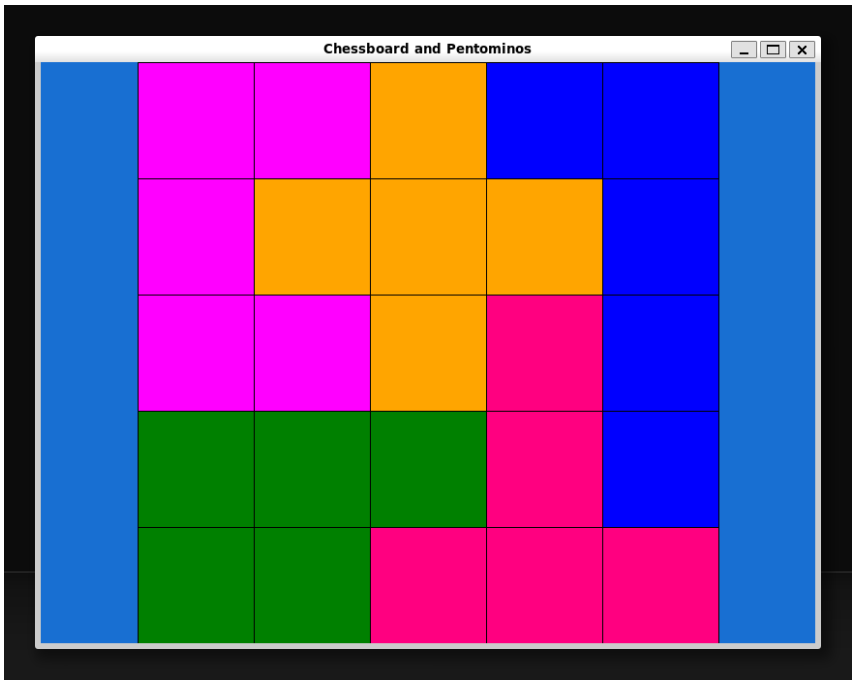

FIGURE 3 – Rendu coloré avec pièces de type Pentominos.

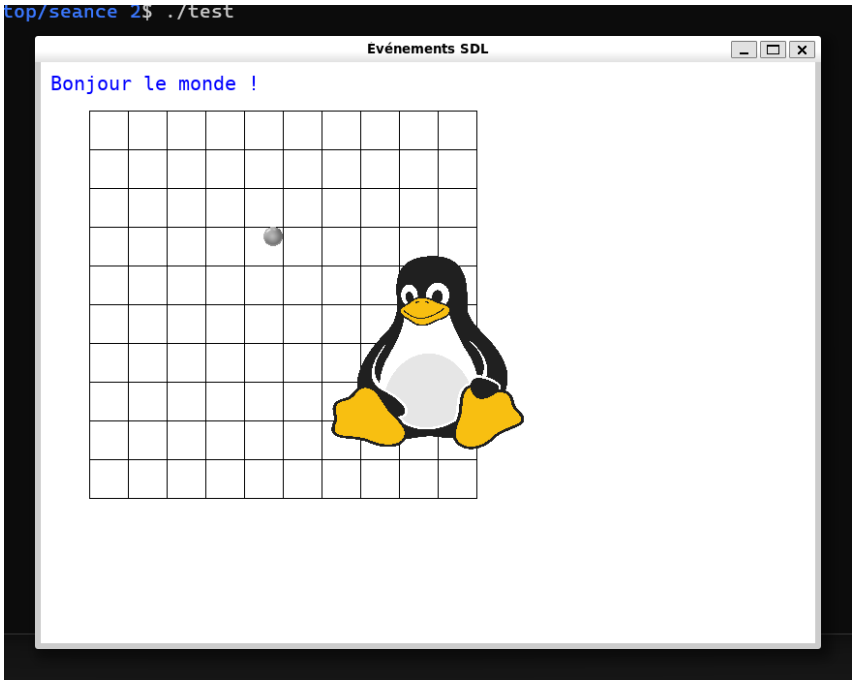

FIGURE 4 – Utilisation de la librairie SDL pour afficher une image (Tux) avec message et grille.