TP Noté SMP 11 — Implémentation du jeu Ricochet Robots

Projet réalisé à Centrale Nantes. Date de rendu prévue : **14 mai 2025**.

Équipe composée de quatre membres : Yanis Cousseau - Simon Cau - Philippe Ocloo - Quentin Balloy

Encadrée par Myriam Servières.

Sommaire

Introduction — Présentation générale du projet et de l'équipe.

Analyse Fonctionnelle — Étude des règles et du fonctionnement du jeu.

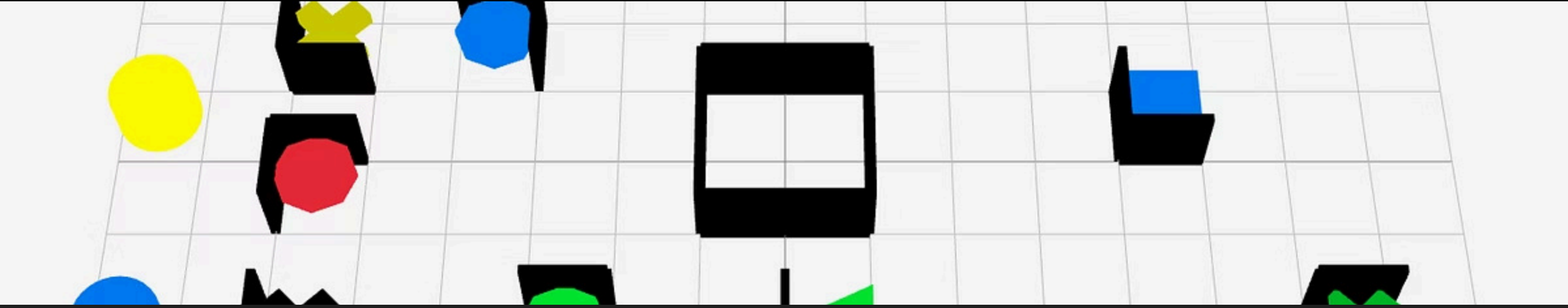
Architecture Logicielle — Structure du système et diagrammes clés.

Conception & Algorithmes — Détail des algorithmes et composants logiciels.

Implémentation & Démonstration — Contraintes pour lisibilité.

Tests et Perspectives — Validation, retours, et évolutions futures.

Made with GAMMA



Introduction au Projet

Objectifs pédagogiques

Approfondir les notions de POO, encapsulation, pointeurs en C++.

Présentation du jeu

Ricochet Robots, un jeu combinatoire stratégique avec déplacements contraints.

Répartition des tâches

Organisation claire au sein de l'équipe.

Made with GAMMA

Analyse Fonctionnelle du Jeu

Règles essentielles

- Robots se déplacent jusqu'à un obstacle ;
- Cibles à atteindre par ordre ;
- Chaque joueur annonce une solution (chemin critique) ;
- Le joueur ayant le moins de déplacements gagne.

Contraintes de simulation

- Déplacements contraints ;
- Génération aléatoire du plateau.

Made with GAMMA

Architecture Logicielle

Diagramme de classes

- Board, Cell, Robot, Player, Target, GameSupervisor ;
- Relations composition et agrégation ;
- Gestion des enums.

Diagramme de cas d’utilisation d'une manche

- Le diagramme représente les interactions possibles entre un joueur et le système au cours d’une manche typique.

Diagramme d'état du jeu

- Expose le comportement du programme. Transitions Front-End, actions Back-End.

Made with GAMMA

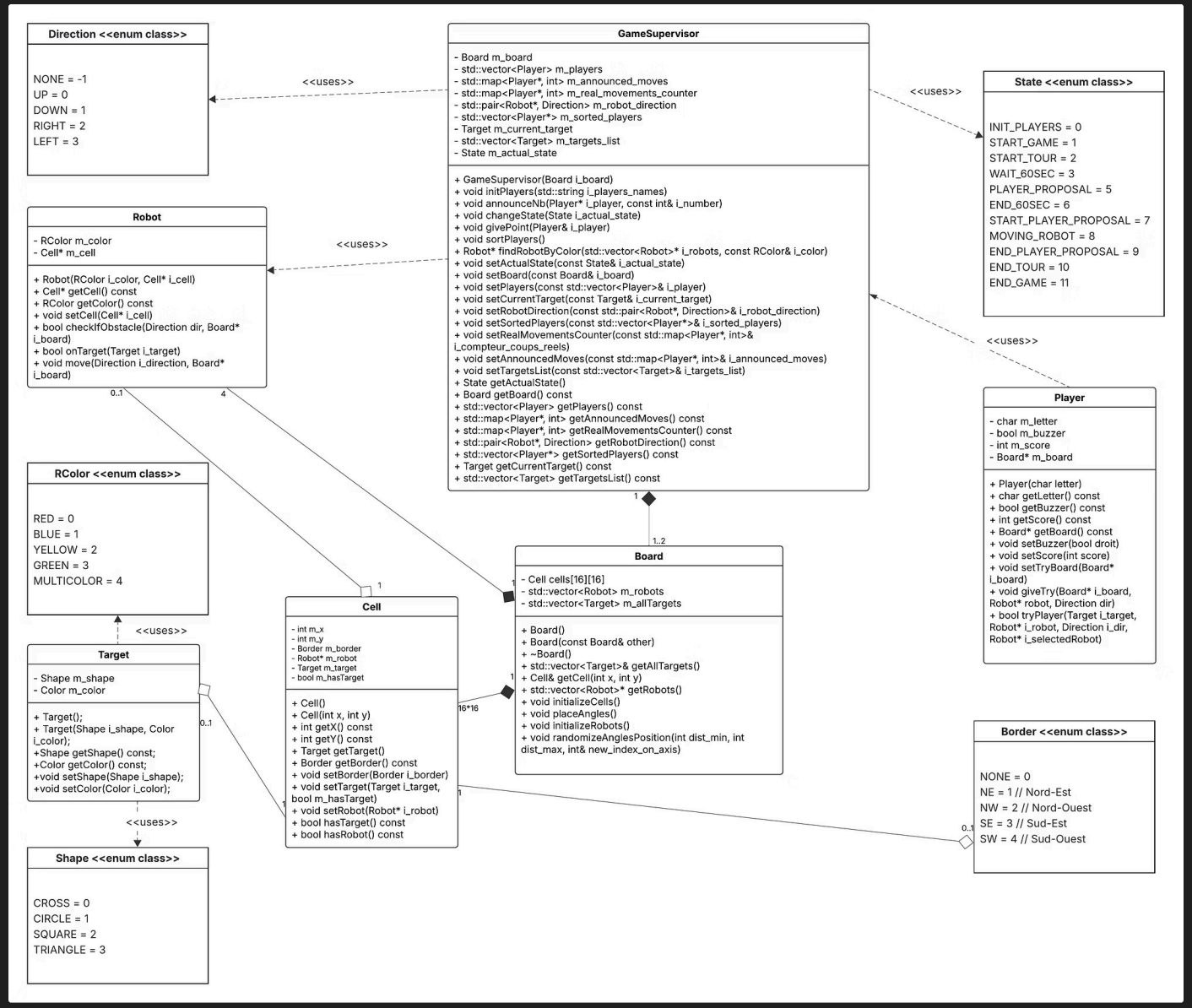


Figure 1 - Diagramme de classe

Made with GAMMA

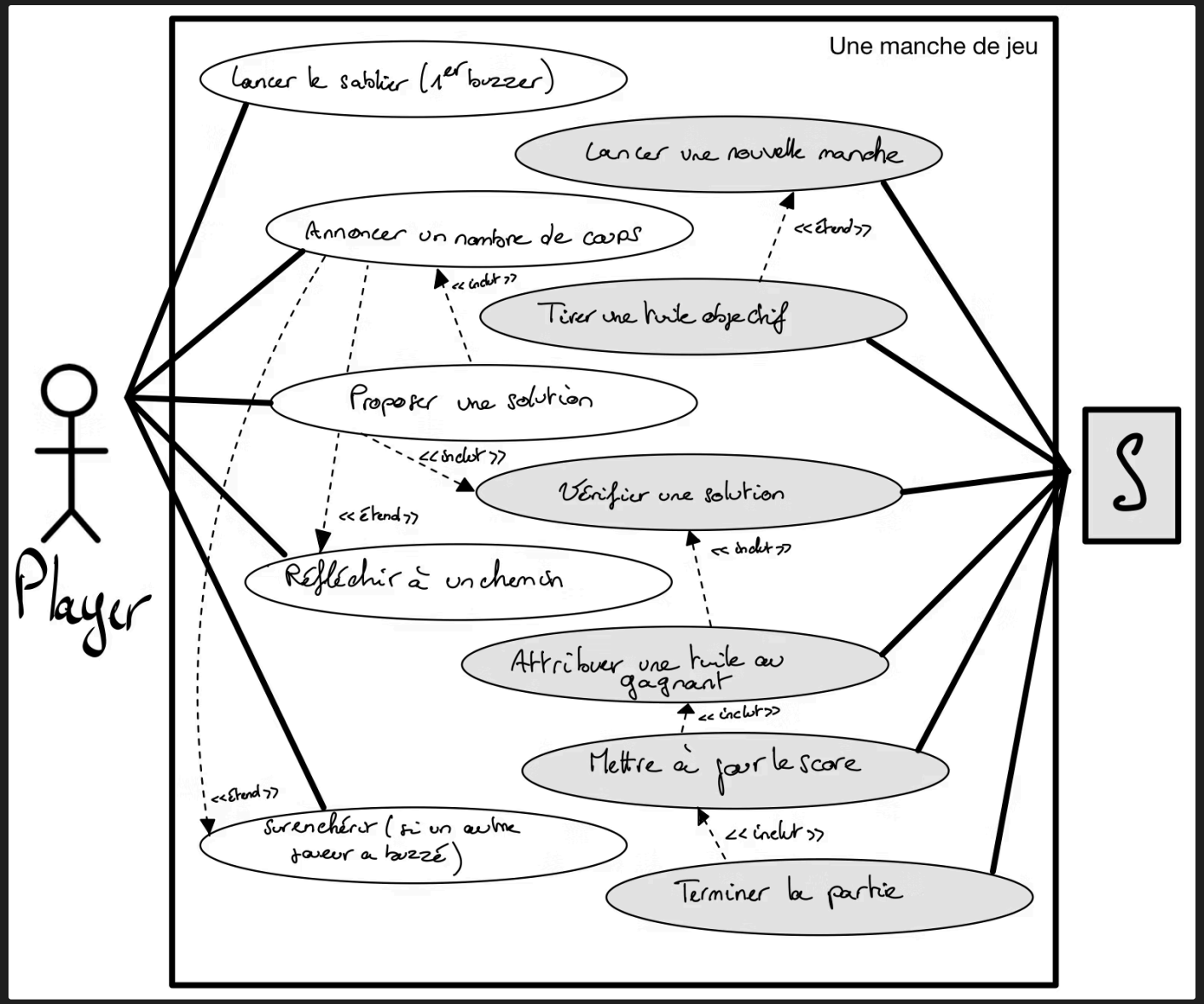


Figure 2 - Diagramme de cas d'utilisation d'un joueur et du système lors d'une manche de jeu.

Made with GAMMA

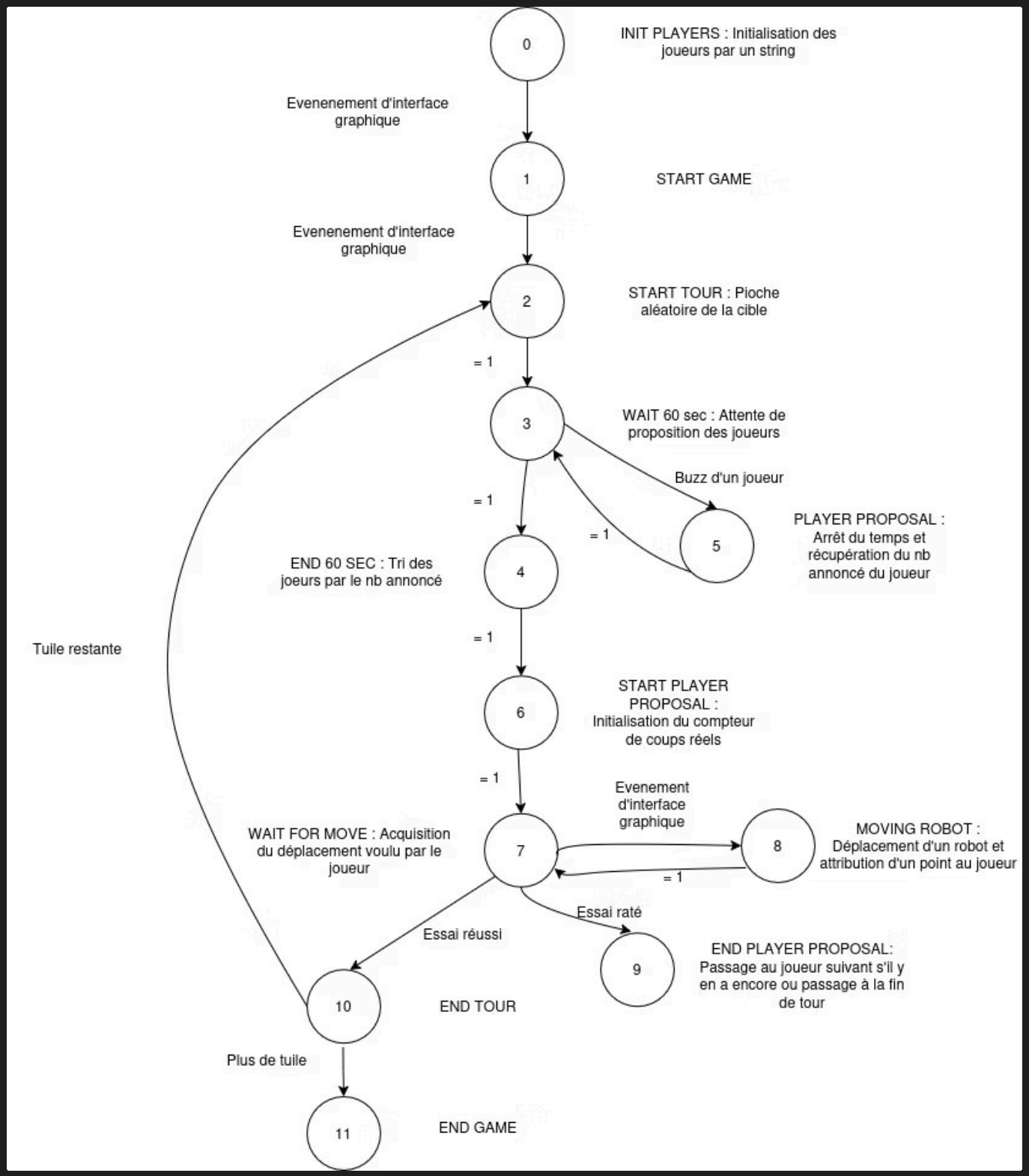
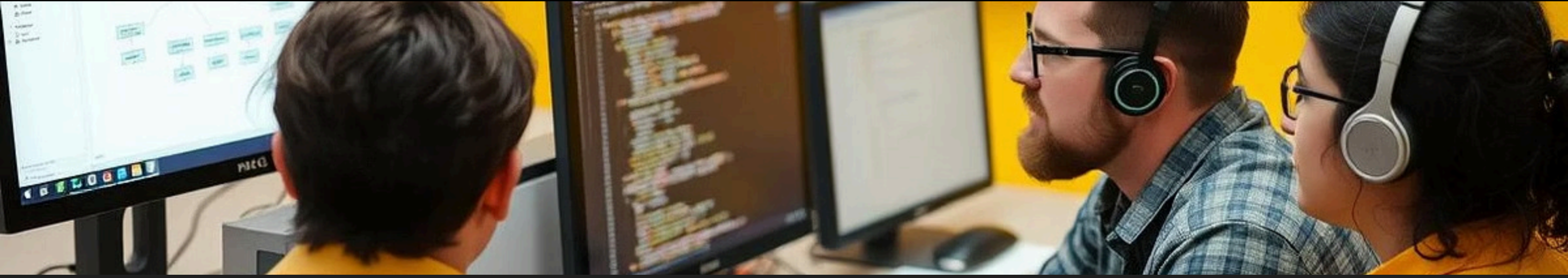


Figure 3 - Machine à états pour le fonctionnement du jeu

Made with GAMMA



Répartition des Tâches

Nous avons créé un Github Project, basé sur un tableau Kanban : **suivre le lien**

Yanis

Interface graphique, CMake, gestion multi-plateforme.

Quentin

UML : diagrammes, classes Board, Cell, Target, tests unitaires.

Simon

Classes Robot, Player, débug transversal et génération du doxygen.

Philippe

Conception machine à états, classe GameSupervisor, tests unitaires.

Made with GAMMA

Conception Détaillée

Génération du plateau

Murs placés selon règles, angles protégés, quarts de plateau modulaires.

Déplacement des robots

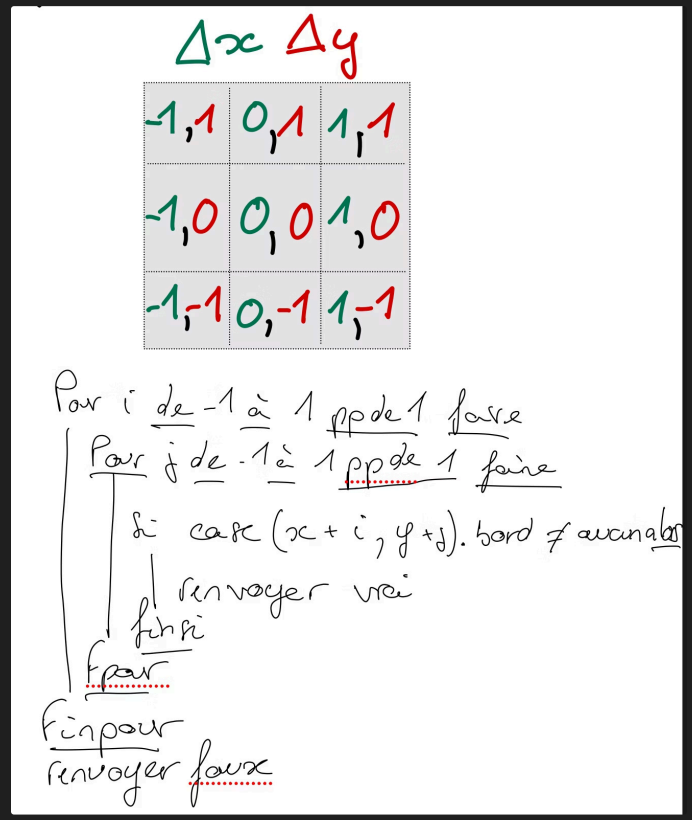
Fonction move() dirige les robots jusqu'aux obstacles. Collision gérée précisément.

Gestion des cibles

17 cibles générées, sélectionnées, attribuant la victoire au joueur gagnant.

Made with GAMMA

Quelques algorithmes...



Principe de rejet des coordonnées (aléatoires) si adjacence d'angle.

Made with GAMMA

Algorithme de la fonction TrouveRobotParCouleur.

O(n)

```
fonction findRobotByColor : Robot*
Problème : Trouver un robot selon sa couleur
Spécifications :
   Paramètres :
      - liste de robots i_robots
      - couleur i_color
   Résultat : pointeur sur robot correspondant (ou nullptr)

Début
   pour chaque robot dans i_robots faire
      si (robot.couleur = i_color) alors
         renvoyer adresse de robot
      fin si
   fin pour
   renvoyer nullptr
Fin
```

Made with GAMMA

```
fonction announceNb : rien
Problème : Enregistrer le nombre de coups annoncé par un joueur
Spécifications :
    Paramètres :
        - pointeur joueur i_player
        - entier i_number
    Résultat : aucun

Début
    m_announced_moves[i_player] ← i_number
    i_player.buzzer ← faux
Fin
```

Algorithme de la fonction annonceNombredeCoups.

O(log(n)) → STD::Map

Made with GAMMA

Algorithme de la fonction triDesJoueurs.

(O(n log n) → STD::Sort)

```
fonction sortPlayers : rien
Problème : Trier les joueurs selon leur nombre de coups annoncés
Spécifications :
   Paramètres : aucun
   Résultat : aucun

Début
   vider m_sorted_players

   créer vecteur vec à partir de m_announced_moves (clé = joueur, valeur =
nb coups)

   trier vec par ordre croissant de valeur

   pour chaque paire (joueur, nombre) dans vec faire
      ajouter joueur à m_sorted_players
   fin pour
Fin
```

Made with GAMMA

```
fonction move : rien
Problème : Déplacer le robot dans une direction jusqu'à un obstacle
Spécifications :
   Paramètres :
      - direction i_direction
      - plateau i_board
   Résultat : aucun

Début
   tant que (pas d'obstacle dans la direction i_direction) faire
      x ← position X actuelle du robot
      y ← position Y actuelle du robot

      selon i_direction faire
         cas UP    : y ← y + 1
         cas DOWN  : y ← y - 1
         cas LEFT  : x ← x - 1
         cas RIGHT : x ← x + 1
      fin selon

      si (x ou y en dehors du plateau) alors
         arrêter la boucle
      fin si

      changer la cellule actuelle du robot vers (x, y)
   fin tant que
Fin
```

Algorithme de la fonction déplacement de robot.

O(1)

Made with GAMMA

Implémentation Technique

Fichiers sources

- .h pour interfaces, .cpp pour implémentations ;
- Compilation CMake ;
- Génération de doc avec Doxygen.
- Séparation des projets "jeu" et interface graphique

Normes de codage

- CamelCase ;
- Commentaires clés ;
- préfixe de variables :
  - l_ = local ;
  - m_ = membre de classe ;
  - i_ = input.

Gestion du dépôt Git

- Forks personnels du dépôt commun ;
- Système de PR pour Revue Par les Pairs (RPP) ;
- Branches structurées et README complet.

Made with GAMMA

GameSupervisor Class Reference

Classe qui supervise le jeu, gère les états, les joueurs, les robots et les cibles. More...

`#include <GameSupervisor.h>`

Public Member Functions

	GameSupervisor () Constructeur de **GameSupervisor** Initialise un nouveau plateau, place les cibles et prépare le jeu.
void	**setBoard** (**Board** i_board) Définit un plateau de jeu.
void	**initPlayers** (std::string i_players_names) Initialise les joueurs à partir d'une chaîne de caractères.
void	**announceNb** (**Player** *i_player, const int &i_number) Enregistre le nombre de coups annoncés par un joueur.
void	**changeState** (enum State i_actual_state) Gère le changement d'état du jeu.
void	**givePoint** (**Player** &i_player) Attribue un point au joueur.
void	**sortPlayers** () Trie les joueurs selon les coups annoncés (ordre croissant)
Robot *	**findRobotByColor** (std::vector< **Robot** > *i_robots, const RColor &i_color) Trouve un robot selon sa couleur.
void	**setActualState** (const enum State &i_actual_state) Définit l'état actuel du jeu.

Figure 4 - Extrait de la doc : classe GameSupervisor

Made with GAMMA

Tests et Validation

Initialisations

Tests de bon fonctionnement des mouvements.

Mouvements testés

Tests manuels et unitaires avec GoogleTest.

Tests fonctionnels

Obstacles, fin de partie et conditions vérifiées.

Made with GAMMA

```
/Users/mac/Desktop/TP11/build> ./unit_tests
Running main() from /Users/mac/Desktop/TP11/build/_deps/googletest-src/googletest/src/gtest_main.cc
[==========] Running 11 tests from 4 test suites.
[----------] Global test environment set-up.
[----------] 2 tests from BoardTest
[ RUN      ] BoardTest.InitializeCells
[       OK ] BoardTest.InitializeCells (0 ms)
[ RUN      ] BoardTest.PlaceAngles
[       OK ] BoardTest.PlaceAngles (0 ms)
[----------] 2 tests from BoardTest (0 ms total)

[----------] 5 tests from GameSupervisorTest
[ RUN      ] GameSupervisorTest.initPlayersTest
[       OK ] GameSupervisorTest.initPlayersTest (0 ms)
[ RUN      ] GameSupervisorTest.FindRobotByColorTest
[       OK ] GameSupervisorTest.FindRobotByColorTest (0 ms)
[ RUN      ] GameSupervisorTest.GivePointTest
[       OK ] GameSupervisorTest.GivePointTest (0 ms)
[ RUN      ] GameSupervisorTest.SortPlayersTest
[       OK ] GameSupervisorTest.SortPlayersTest (0 ms)
[ RUN      ] GameSupervisorTest.SettersAndGettersTest
[       OK ] GameSupervisorTest.SettersAndGettersTest (0 ms)
[----------] 5 tests from GameSupervisorTest (0 ms total)

[----------] 2 tests from PlayerTest
[ RUN      ] PlayerTest.ConstructorTest
[       OK ] PlayerTest.ConstructorTest (0 ms)
[ RUN      ] PlayerTest.SettersAndGettersTest
[       OK ] PlayerTest.SettersAndGettersTest (0 ms)
[----------] 2 tests from PlayerTest (0 ms total)

[----------] 2 tests from RobotTest
[ RUN      ] RobotTest.ConstructorTest
[       OK ] RobotTest.ConstructorTest (0 ms)
[ RUN      ] RobotTest.MovementDirectionTest
[       OK ] RobotTest.MovementDirectionTest (0 ms)
[----------] 2 tests from RobotTest (0 ms total)

[----------] Global test environment tear-down
[==========] 11 tests from 4 test suites ran. (0 ms total)
[  PASSED  ] 11 tests.
(base) ▓▓▓ mac@MacsterRace ▓▓▓▓ Mer mai 14 10:13:15P
/Users/mac/Desktop/TP11/build>
```

Figure 5 - Résultat de tests unitaires

Made with GAMMA

Retours et Perspectives

Difficultés

- Gestion collisions.
- Génération aléatoire.
- Affichage dynamique.

Points forts

- Modularité.
- Structure du projet.
- Lisibilité du code.

Améliorations futures

- Gestion du cas "victoire en un coup".
- Refactoring des classes.
- IA optimisant trajets (proposition de solution parfaite).
- Intégration d'éléments 3D dans l'IG.
- Utilisation d'un design pattern spécifique (ex: MVC).
- Implémentation d'une Intégration Continue pour évolution avec tests de no-reg.

Made with GAMMA

Conclusion du Projet

Bilan technique et humain

Excellente acquisition des compétences objets et modularité.

Gestion de projet

Collaboration efficace et organisation rigoureuse des tâches. Partage riche entre les élèves.

Introduction à l'architecture

Projet formateur pour concevoir un logiciel. Ouvre sur les enjeux des design patterns et leur intérêt dans un contexte similaire.

Made with GAMMA