Project 3 - PACoronam

Alexandre Baré - 20172388
Boris Courtoy - 20175068

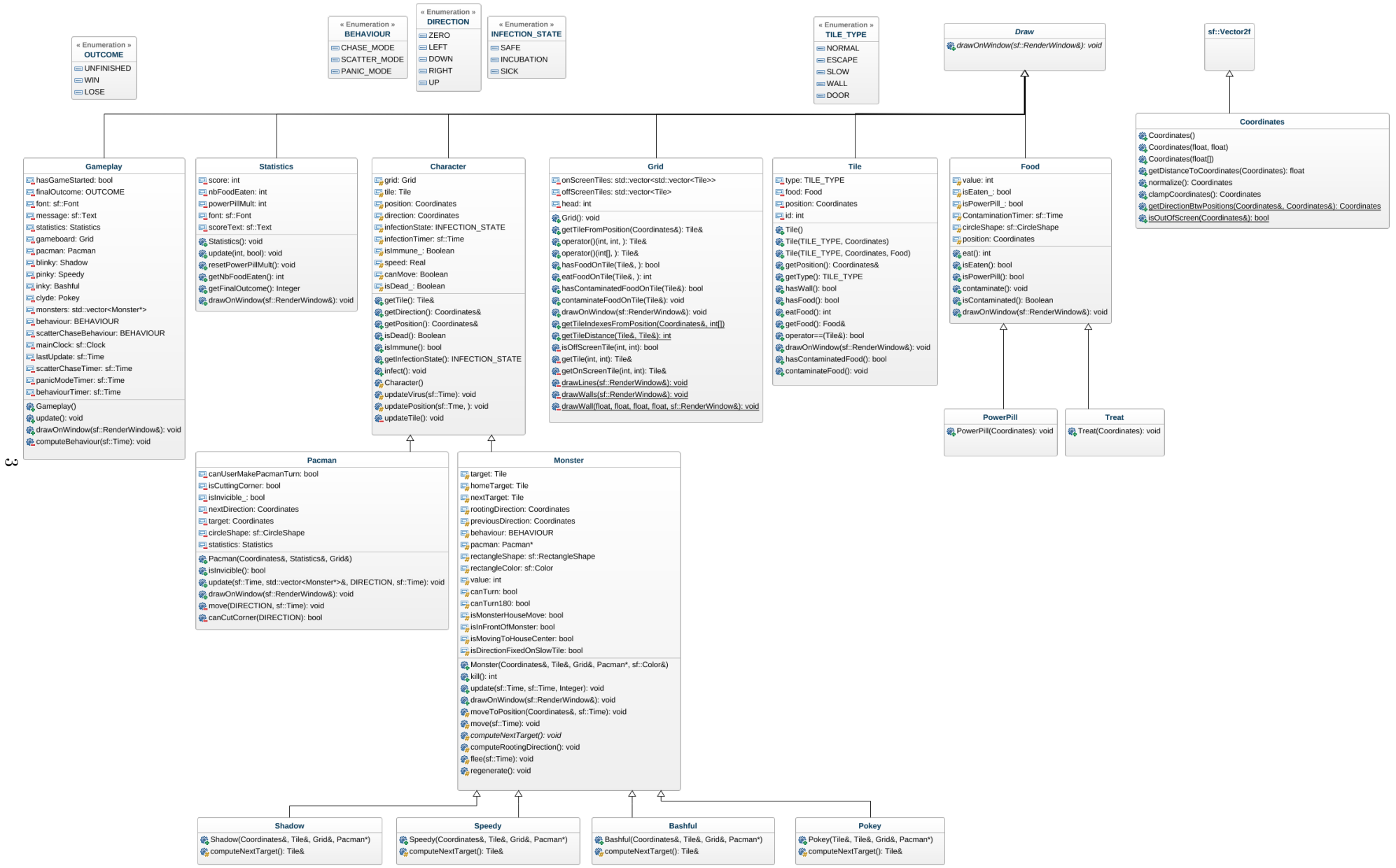
March 2020

Design and architecture of the code

Our implementation is composed of 15 classes splitted into 16 cpp files and 17 hpp files. For more details, see the class diagram. It is good to note that there are also different enum classes visible on the diagram. Here is the utility of each class :

- Gameplay is the main class of the game. It contains everything needed during the game and all the function needed to run a game correctly. Its instantiation also creates all the objects needed. It is responsible for the display of the game objects, the collision management, the computation of the global scatter-chase cycle and also the panic mode, the user inputs management, the time keeping, the propagation of the virus and the display of the messages in the center of the screen.
- Statistics is a class containing the score, the number of food eaten, the power pill multiplier (i.e.: the multiplier of the score when monsters are being eaten in a row) and the score label. It is a general purpose class to manage the different "statistics" of the game.
- Coordinates is inheriting from sf::Vector2f and is used to localise the objects of the game, or to represent directions or general-purpose vectors. We chose to use such a class instead of sf::Vector2f so as to add different useful operations on 2D vectors.
- Draw is an abstract class that is the parent of every drawable object in the game. It is used to ease the drawing of each object in the grid by imposing on its children to implement the drawOnWindow function.
- Grid is the maze. It is composed of 2 vector of tiles used to represent all the tiles. The first vector is used to represent the tiles that are visible while playing and the second one represent a temporary buffer for the tiles that aren't visible but that are used for the target tiles of the monsters and for going through the underground tunnel.
- Tile represents a cell of the grid. Each tile is identified by its coordinates, by its type and by the food it contains (or not). We used 5 different tile types : normal for the basic tiles, escape for the 4 yellow tiles where monsters can't go north, slow for the 5 cells before each extremity of the tunnel, wall for the tiles containing a wall or a door. Each tile can have a food that is either a treat or a powerpill.
- Food is a class managing the data contained in each tile. Each food has a score value, a position and 3 boolean telling if the food is eaten, is a powerpill or is infected. Food is the parent of Treat and PowerPill.
- Treat and PowerPill inherits from Food, they are just used for the 2 different graphical representations and score values of the food pieces in the maze.
- Character is the class containing the characteristics needed for a movable character such as its speed, its direction, its coordinates and also the management of the virus on those characters. Character is the parent of Monster and Pacman.
- Pacman is the class used for the main character of the game. It inherits from Character and have some new attributes such as its possible invicibility. This class also contains a bunch of new methods used to control pacman properly in the maze and the graphical design.
- Monster is the class used for the 4 types of monster. It inherits from Character but has also new features. The features added are some tile references for the routing, the behaviour they have and their score value. This class also contains all the needed function for the monsters to compute their next position and interact with the different maze components. It also contains the graphical design of a monster.
- Bashful, Pokey, Shadow and Speedy are children of Monster. They are used for their specific graphical design and for the computation of the target tile in chase mode and scatter mode for each monster since they only act differently by the computation of their target tile.

Our two non-class files are constants.hpp and main.cpp.
Constants.hpp is used to initialize global constants such as the FPS number or the size of a cell.
Main.cpp is used for launching and updating the game and for the management of the window and the reset function. First of all, the window is created. After this, we enter a loop and create the gameplay object. We create this object in a loop so that we can easily reset the game by creating a new gameplay object. Then, as long as the window is opened and the R key is not pressed, the gameplay object is updated and displayed 60 times per second.


« Enumeration »
OUTCOME
UNFINISHED
WIN
LOSE
« Enumeration »
BEHAVIOUR
CHASE_MODE
SCATTER_MODE
PANIC_MODE
« Enumeration »
DIRECTION
ZERO
LEFT
DOWN
RIGHT
UP
« Enumeration »
INFECTION_STATE
SAFE
INCUBATION
SICK
« Enumeration »
TILE_TYPE
NORMAL
ESCAPE
SLOW
WALL
DOOR
Draw
drawOnWindow(sf::RenderWindow&): void
sf::Vector2f
Coordinates
Coordinates()
Coordinates(float, float)
Coordinates(float[])
getDistanceToCoordinates(Coordinates): float
normalize(): Coordinates
clampCoordinates(): Coordinates
getDirectionBtwPositions(Coordinates&, Coordinates&): Coordinates
isOutOfScreen(Coordinates&): bool
Gameplay
hasGameStarted: bool
finalOutcome: OUTCOME
font: sf::Font
message: sf::Text
statistics: Statistics
gameboard: Grid
pacman: Pacman
blinky: Shadow
pinky: Speedy
inky: Bashful
clyde: Pokey
monsters: std::vector<Monster*>
behaviour: BEHAVIOUR
scatterChaseBehaviour: BEHAVIOUR
mainClock: sf::Clock
lastUpdate: sf::Time
scatterChaseTimer: sf::Time
panicModeTimer: sf::Time
behaviourTimer: sf::Time
Gameplay()
update(): void
drawOnWindow(sf::RenderWindow&): void
computeBehaviour(sf::Time): void
Statistics
score: int
nbFoodEaten: int
powerPillMult: int
font: sf::Font
scoreText: sf::Text
Statistics(): void
update(int, bool): void
resetPowerPillMult(): void
getNbFoodEaten(): int
getFinalOutcome(): Integer
drawOnWindow(sf::RenderWindow&): void
Character
grid: Grid
tile: Tile
position: Coordinates
direction: Coordinates
infectionState: INFECTION_STATE
infectionTimer: sf::Time
isImmune_: Boolean
speed: Real
canMove: Boolean
isDead_: Boolean
getTile(): Tile&
getDirection(): Coordinates&
getPosition(): Coordinates&
isDead(): Boolean
isImmune(): bool
getInfectionState(): INFECTION_STATE
infect(): void
Character()
updateVirus(sf::Time): void
updatePosition(sf::Tme,): void
updateTile(): void
Grid
onScreenTiles: std::vector<std::vector<Tile>>
offScreenTiles: std::vector<Tile>
head: int
Grid(): void
getTileFromPosition(Coordinates&): Tile&
operator()(int, int,): Tile&
operator()(int[],): Tile&
hasFoodOnTile(Tile&,): bool
eatFoodOnTile(Tile&,): int
hasContaminatedFoodOnTile(Tile&): bool
contaminateFoodOnTile(Tile&): void
drawOnWindow(sf::RenderWindow&): void
getTileIndexesFromPosition(Coordinates&, int[])
getTileDistance(Tile&, Tile&): int
isOffScreenTile(int, int): bool
getTile(int, int): Tile&
getOnScreenTile(int, int): Tile&
drawLines(sf::RenderWindow&): void
drawWalls(sf::RenderWindow&): void
drawWall(float, float, float, float, sf::RenderWindow&): void
Tile
type: TILE_TYPE
food: Food
position: Coordinates
id: int
Tile()
Tile(TILE_TYPE, Coordinates)
Tile(TILE_TYPE, Coordinates, Food)
getPosition(): Coordinates&
getType(): TILE_TYPE
hasWall(): bool
hasFood(): bool
eatFood(): int
getFood(): Food&
operator==(Tile&): bool
drawOnWindow(sf::RenderWindow&): void
hasContaminatedFood(): bool
contaminateFood(): void
Food
value: int
isEaten_: bool
isPowerPill_: bool
ContaminationTimer: sf::Time
circleShape: sf::CircleShape
position: Coordinates
eat(): int
isEaten(): bool
isPowerPill(): bool
contaminate(): void
isContaminated(): Boolean
drawOnWindow(sf::RenderWindow&): void
PowerPill
PowerPill(Coordinates): void
Treat
Treat(Coordinates): void
Pacman
canUserMakePacmanTurn: bool
isCuttingCorner: bool
isInvicible_: bool
nextDirection: Coordinates
target: Coordinates
circleShape: sf::CircleShape
statistics: Statistics
Pacman(Coordinates&, Statistics&, Grid&)
isInvicible(): bool
update(sf::Time, std::vector<Monster*>&, DIRECTION, sf::Time): void
drawOnWindow(sf::RenderWindow&): void
move(DIRECTION, sf::Time): void
canCutCorner(DIRECTION): bool
Monster
target: Tile
homeTarget: Tile
nextTarget: Tile
rootingDirection: Coordinates
previousDirection: Coordinates
behaviour: BEHAVIOUR
pacman: Pacman*
rectangleShape: sf::RectangleShape
rectangleColor: sf::Color
value: int
canTurn: bool
canTurn180: bool
isMonsterHouseMove: bool
isInFrontOfMonster: bool
isMovingToHouseCenter: bool
isDirectionFixedOnSlowTile: bool
Monster(Coordinates&, Tile&, Grid&, Pacman*, sf::Color&)
kill(): int
update(sf::Time, sf::Time, Integer): void
drawOnWindow(sf::RenderWindow&): void
moveToPosition(Coordinates&, sf::Time): void
move(sf::Time): void
computeNextTarget(): void
computeRootingDirection(): void
flee(sf::Time): void
regenerate(): void
Shadow
Shadow(Coordinates&, Tile&, Grid&, Pacman*)
computeNextTarget(): Tile&
Speedy
Speedy(Coordinates&, Tile&, Grid&, Pacman*)
computeNextTarget(): Tile&
Bashful
Bashful(Coordinates&, Tile&, Grid&, Pacman*)
computeNextTarget(): Tile&
Pokey
Pokey(Tile&, Tile&, Grid&, Pacman*)
computeNextTarget(): Tile&