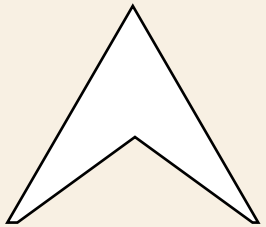
CIPHER ROGUE

Réalisé par Kévin Angelier-Leplan
Supervisé par Élodie Desserée

ENTITÉS DU JEU

Player

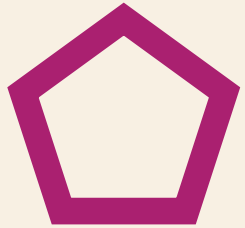


Shooter
Sniper

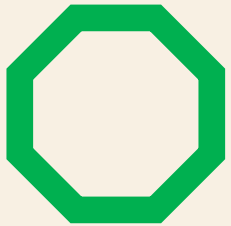
Bomber

Seeker

Charger



Turret



Spawner

Bullet
Bombshell

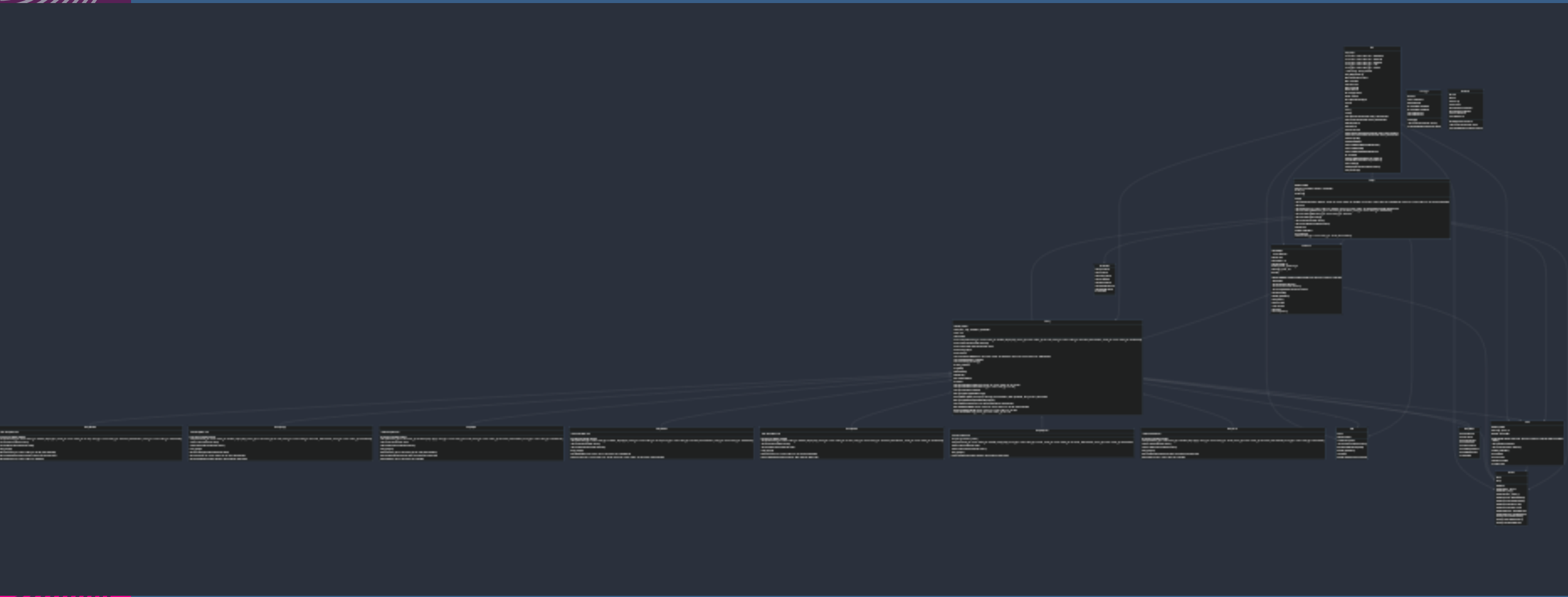
Bullet

Bombshell

Bullet

Seeker

DIAGRAMME DES CLASSES



https://mermaid.ink/svg/pako:eNrtW_9v2zYW_1d4KhA4s72Lt8vaBm2AJm2WAQtaNMYV2OVwYCza5iKJmkQl8XLt3z5-1yP1JY6bQ7er9UMs8X3h4-PnPT5SyI00YzGJDqJZgsvyNcWLAqcXGRLXm4ykK_Tiv-Oxvj1fMsZJ0U48XuJi0UU8z2jeRTti6WWnHCFXnbQc32RdxGlVFlRfZBfZuwSvSlEsQTb9iFMiGg6RpoEGw-KeP-AkAY9HVZJlrXWDsL1cEo_pHSsppyyTPWmzak31s1VVt9RydZseKMe8lNpcb2A0sM3vW_fQaAYesVaBBmsWbHI9uOZAKWDWt85iPTkKWUg66M6w5vpHk-U1RKXAD4n_k_PiBbomM87Eb5XR3yqi2rRhh4eH6FLdlWb8myt4lSQPkrduUCrsQ_kADdLZUvgGP0xOjVQKEnFDiSdq2Ute0GxxiFJSlnhBpqRlaYaTmnWsOVBMS-X8-ExzAo55wjBvMExpaoMMcM2qomRFSKuyki4yEiOacZSQawlNuGQsMXL_pCW9TEhlVBlCOwUnMaBJZcRF_DErOSDGrBKaxPiFlDQnkEtxfk4S4SQB1FNaCl-t_jXZ-zfgmpJbjrj4A9pOmBAF6JSeVpfE8GAXEizFXJ9aOAlWc10zGqMqjzEnA_SeZDEpPtAsZjc76Eb9jpCZEJJwLIcWaG1XC7XHBb7ZUHevcnOpOaOlDunXBMeDXRSqWUcPNJlmlP8sYdCm6wHKzOVw-Gq2pOLXNxHMuPPXCb5y0bOp78a1-491nDySQiKCmK1kHikHu033wLHoJHfMkkTEGst6-lV2MAGzJZId6VxZOukPlC9d_utT5XR1qXz79qhPPJTvUCNTtzNON6kk2au6QzdASqkyxRnO-_X0K4Im5xWXk3VSsHTKBnbBRHPxPHLrJ-Ksv7u1-oPdzmmSPEa_D-nYm6mCiKx2vx836sFLcHqtUstQe2zda8DmFkBDSrkyeRD86Bc_1uI3YPHWUX-LGk0rZlWHVs6qdQ1oYMRt9lg1OSSmBV3-rGzeA2SzdAoOc-dU1BhhomzHYtUcOhTtlMaXpOhmHXexNji_MWMoZzjB3Wx_b2VDoBdr5MsBQvfZWbOPe9jV2mYZX67L-Ld1GV908gW4kWEMMDPWeYr-TuDiVc9Bbm8ApqSKQYMDBMbwnmlBspo1_6fM18oufxWZUzxC3tqsBeEhkDWHHl0gnosBDXpEBdvAqy2FXeS23WOnLCW5qFuN13QtN1cFXb3mq6qvKgjilCfEPhjaeV5Qbkjm3r8M33sxZJwtBNMS50TtaY4qzoXRHRy_VZRbDmgIEFWPqGEqEDUcFtbisiN2OWKNAhCySocucRYn5Dihs6tegcDZourjS-jtoPCvvQ8Hk8Kth25SMlNXhYeeS0lW6eHf51m_HysFNLe40_rCjWbQqBt3XUZew7OW8zMc66otuGDUrc14HjWTwFQeyHC7PpQ5IbF7EAE30gW8iOOVF3qf6I68dm_D0lOydnoI7TTzSc375jZPmKzn1hAyWei1qNc4FivXwDI93cSjzW9LPY7ySsLLlzJ7aEGcrnKfZkySlWmzfYILNbxYbkDa9h-GjXJ7ItXY4avi4ptDyfPGHgM0duVyWs_wbaN9LiZZzhJ6ifYaRC4lU3YiWAT5e0BuwZgPrYYqFZEjCzVpTbjFx-7QJQS8qulh2nVLG9K9zu5Ht7lVu6dqxuXJg6uGPpleQD6EOO9A-bA7wQK81hM0RB2QhCBogNJHSlhLQB2iOiLqMYbazVr7GqdqrVDEvtlGa0wvapvfkedqBKbb1ML1GaGZ7yGYxBMs8Y58gujKtCOfEqvRtNOWBCd82U6b04K8FzPcTtWb53NgTjtDi1nSy4a6ZHqhbgxdjnpo7iXyht2HQRZOI-3kdc84d-wh55py4GxzBxxuridtzjV39MFm8zRDOWElNkp6zP4GSr5aGHyZMXYsnNZuz86CzAi9JjqIfprLOf4iVq9j3CIdzzZ7vrxjD5jvV64KJB17rcw95VEX-6mK1m4hFAi6jKfOHZv0zoQbMNZJV4Y74GjJiKiZE20yIPhV5xAjuUSbO0U072E8WbuLE8y5TzFbcZVqtRmep1J2TQaPFpTBGqyAcAd8c00LXuEkKP02wbr33me0gf0bwLgx6NHnBJwHWuiX3q1Aj8yb-Vx0Xz5AtKX-bDJ9csdQ66WuzRLX5l70z_F18gcR6oIulzfv6rOKdZMR4OypuIbw1Nevt4ZBfjlj17gupoZd2aczodj6srXf09zmm1A1jn-tSmf2ES5J_DYzhf9go5S-Vg_61cJDo3O3mZhbtavD_1ZXqZp2yvTCW2OAg4X4InvSmFlQqj7xdZq0bCtWkJ1hTgZiaq-eu6r2SfCaK9UwgBQdZ53bE29H4u9VgDWAV9s0cpbUawS0YBfaFvrvpOJVoVNtHVAw83rSOvZSYcxyWhWZ2exon7-CBgulGSiG295XhTblWFTelpOJdBVO5-gRVi-wLCK7w_6ZZuTt_JwuIvzzu_R6sOxW6yHC8i31o_QCNgfgaxi4CMt325zOwLcgqqWELAgKtx72DlvfO2_X8o3r1f7lO5AKF26_svO3QBugqmegQSe2fvwCM9YCdvPN2IZoN9JbuP9V4O6UfcBFJhh-YRnpLH_Hj4Hbxq4n1Cut0e94us5dwhMoMRPHLGEFmsm_u3oXpUs4gGz94elW2F9VHv8rlNVHqf6g9042b4G6BeqfBqgdpdHDwaNQj0LYq6_Rt6jfov7_GfUh6PW_S2xR_zWi_n9RPOh_sdmWuFs0_TlyXZjvFP6-jg8MolGUkiLFNl4OljXki4gvSUouogNxG-Pi6iLSfLji7HyVzaKDOU5KMop0fJn_-3OtJKai1zPzn4Esm9OFEM5xFh3cRbfRwXjv2z1xTZ5-93x_Mpl8P3n-_Nlkf7l_ilbRweQfe99Onu_98N2zvf3J0_2nzz6Oot8ZE9onSscv6l719PEPE7PUbA

ENEMY

```
void Enemy::smoothTurn(float targetAngle, float turnSpeedFactor, float deltaTime){
    float angleDiff = targetAngle - angle;

    // angleDiff stays between PI and -PI
    if (angleDiff > M_PI) {
        angleDiff -= 2 * M_PI;
    } else if (angleDiff < -M_PI) {
        angleDiff += 2 * M_PI;
    }

    angle += angleDiff * turnSpeedFactor * deltaTime * 60;

    // angle stays between PI and -PI
    if (angle > M_PI) {
        angle -= 2 * M_PI;
    } else if (angle < -M_PI) {
        angle += 2 * M_PI;
    }
}
```

ENEMY

```
void Enemy::adjustPositionBasedOnEnemies(std::shared_ptr<std::vector<std::unique_ptr<Enemy>>>& enemies){
    float angleEnemyEnemy, hitBoxBoth, moveDistance;
    Position enemyPos, diffPos;
    for (const auto & enemy : *enemies) {
        if (enemy.get() == this) continue; //We skip if it's the enemy itself

        enemyPos = enemy->getPosition();
        hitBoxBoth = static_cast<float>(enemy->getSize() + size);
        diffPos = position - enemyPos;
        // sqrt(x² + y²) < distance <=> x² + y² < distance²
        if (diffPos.x * diffPos.x + diffPos.y * diffPos.y < hitBoxBoth * hitBoxBoth){
            angleEnemyEnemy = std::atan2(enemyPos.y - position.y, position.x - enemyPos.x);
            moveDistance = hitBoxBoth - std::sqrt(diffPos.x * diffPos.x + diffPos.y * diffPos.y);

            if(enemy->isMovable()){
                enemy->setPosition(enemyPos + Position(std::cos(M_PI+angleEnemyEnemy) * moveDistance / 4,
                                                       - std::sin(M_PI + angleEnemyEnemy) * moveDistance / 4));

                position += Position(std::cos(angleEnemyEnemy) * moveDistance / 4,
                                     - std::sin(angleEnemyEnemy) * moveDistance / 4);
            }
            else{
                position += Position(std::cos(angleEnemyEnemy) * moveDistance / 2,
                                     - std::sin(angleEnemyEnemy) * moveDistance / 2);
            }
        }
    }
}
```

PLAYER

```cpp
void Player::move(float deltaTime) {
    int xAxisMove, yAxisMove = 0;
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) { yAxisMove++; }
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) { xAxisMove--; }
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) { yAxisMove--; }
    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) { xAxisMove++; }

    if (yAxisMove != 0 && xAxisMove != 0) {
        position += Position(xAxisMove * std::sqrt(2)/2,-yAxisMove * sqrt(2)/2) * speed * deltaTime * 60;
    } else {
        position += Position(xAxisMove,-yAxisMove) * speed * deltaTime * 60;
    }
}
```

PLAYER & ENEMY

```
for (const auto & wall : *walls) {
    Position wallPos = wall->getPosition();
    float angleWallPlayer = std::atan2(wallPos.y - position.y, position.x - wallPos.x);

    if(wall->isInWall(position + Position(-std::cos(angleWallPlayer),std::sin(angleWallPlayer))*size)){

        if (-M_PI/4 <= angleWallPlayer && angleWallPlayer <= M_PI/4) {
            position.x = wall->getPoint(1).x;
        } else if (angleWallPlayer >= M_PI*3/4 || angleWallPlayer <= -M_PI*3/4) {
            position.x = wall->getPoint(0).x;
        } else if (M_PI/4 <= angleWallPlayer && angleWallPlayer <= M_PI*3/4) {
            position.y = wall->getPoint(1).y;
        } else {
            position.y = wall->getPoint(2).y
        }

    }
}
```

GAME

```cpp
int Game::selectMap() {
    std::vector<double> weights;

    for (int i = 0; i < 10; i++) {
        double weight = level - mapSelectionHistory[i];
        weights.push_back(weight);
    }

    std::random_device rd;
    std::mt19937 gen(rd());
    std::discrete_distribution<> dist(weights.begin(), weights.end());

    int selectedIndex = dist(gen);
    mapSelectionHistory[selectedIndex] += 1;
    return selectedIndex;
}
```

GAME

```
void Game::update(sf::RenderWindow &window, float deltaTime) {
    //...
    if (enemies->size() == 1) {
        for (unsigned int i = 0; i < bulletsAlly->size(); i++) {
            if (isClose(*bulletsAlly->at(i), *enemies->at(0))
                && bulletsAlly->at(i)->getDamage() >= enemies->at(0)->getHp()) {
                    deltaTime /= 5;
                    break;
            }
        }
    }
    //...
}
```

CONCLUSION

Défis :

- Collisions entre les ennemis et avec les murs
- Gestion de la mémoire

Amélioration :

- Plus de type d'ennemis
- Plus de type de projectiles
- Plus d'obstacles
- Des améliorations entre chaque niveau
- Des boss
- Effets visuels
- Meilleure gestion des déplacements