PROJET LONG

DoomMetal

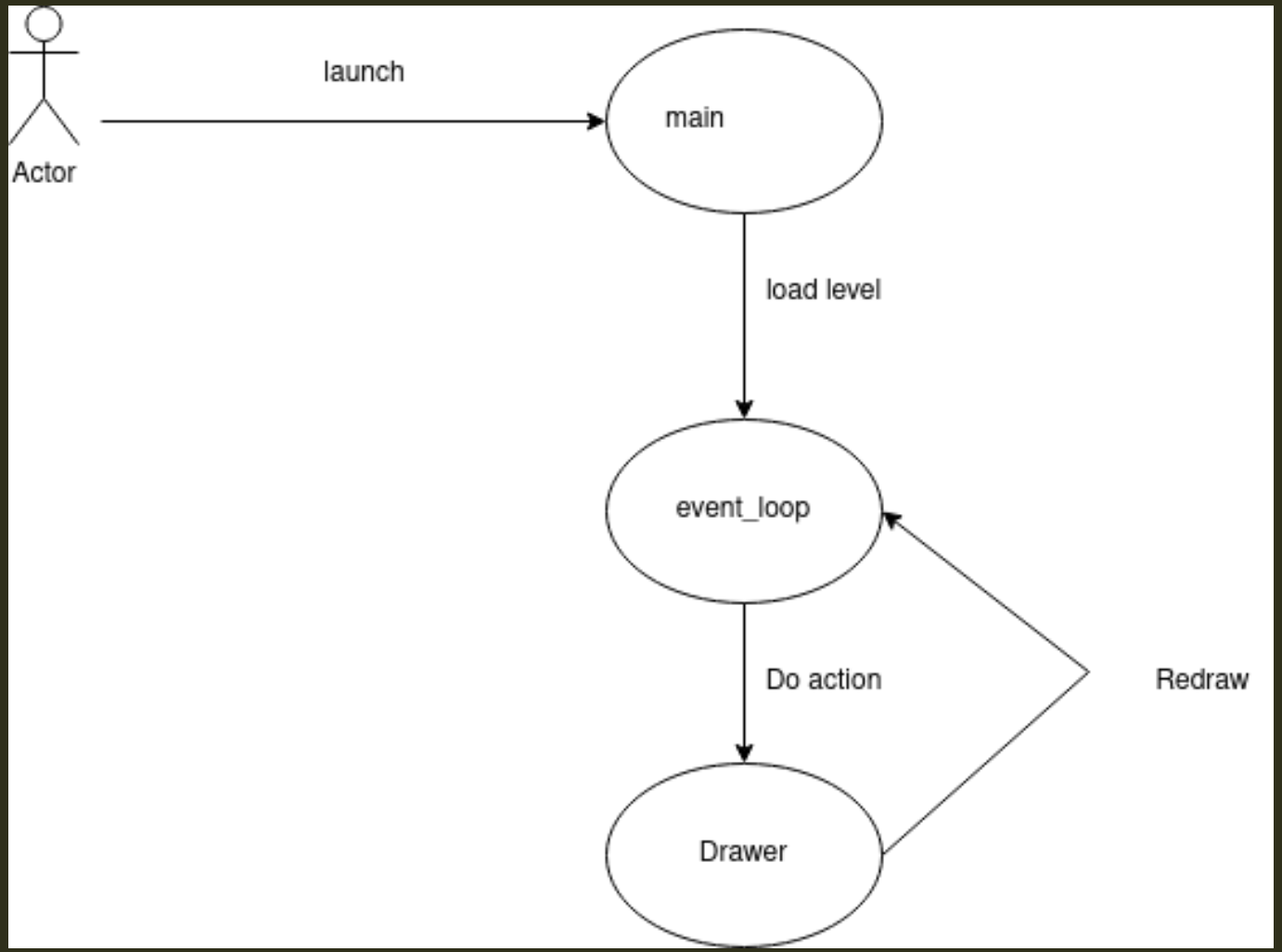
Par Tazouev Arbi et Guinet Virgile

OBJECTIF / PROBLEMATIQUE

- Peut-on créer un jeu vidéo en Ocaml qui utilise la même technique de rendu 3D que Wolfenstein 3D de 1992?

DEMO LIVE

ARCHITECTURE DU PROJET



- Lancement du programme
- Choix d'un niveau
- Traitement des entrées d'utilisateur
- Rendu du jeu sur l'écran

CHOIX DE CONCEPTION

- Algorithme de Ray casting choisit: DDA (Digital Differential Analysis)
- On utilise des records avec des champs mutable pour faciliter le développement
- La carte du jeu est stocké dans un type array a la place d'une liste pour optimiser l'accès et la modification

ÉTAPES CLÉS

- Choix de la librairie graphique
- Mise en place du code pour l'affichage d'une fenêtre, le dessin dessus et les events
- Création des types avec toute l'information sur le jeu et afficher le niveau en 2D
- Implémentation de l'algorithme de Ray casting et affichage simple en 3D
- Ajout des textures
- Ajout des ennemies et des tirs
- Menu principal et le HUD (affichage de la barre de vie et de l'arme)
- Ajout d'animations

MODULE PRINCIPAUX

- Main.ml - contient la boucle principale du jeu, les events et l'initialisation des ressources du jeu.
- Drawer3D.ml - fait toute la partie de rendu sur l'écran d'utilisateur avec le HUD et les animations
- Entity.ml gère les déplacements des ennemis et du personnage, contient le code pour l'IA des ennemies
- AST.ml contient tous les types utilisés dans ce projet

COMPÉTENCE ET RÉPARTITION DU TRAVAIL

- Nous avons dû apprendre à nous servir de la librairie graphique OcamlSdl2 ainsi que l'apprentissage de l'algorithme de ray casting

Arbi

- Mise en place de la librairie graphique
- Affichage 2D
- Ray casting
- Affichage 3D simple puis texturé
- Menu principal
- Affichage des textes

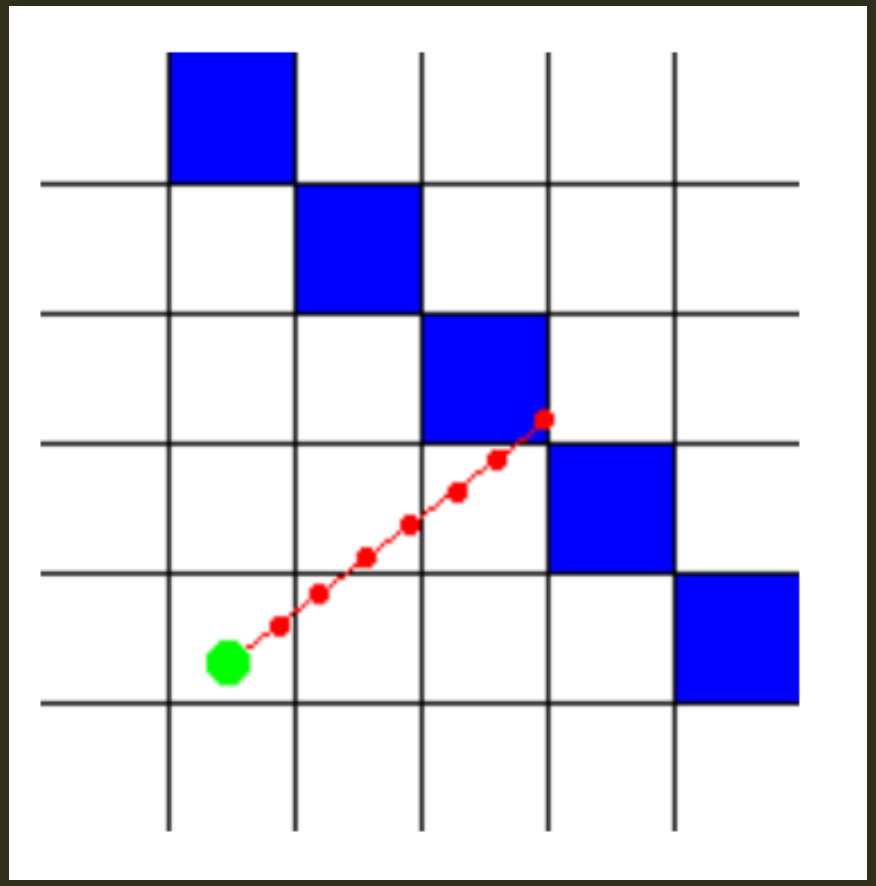
Virgile

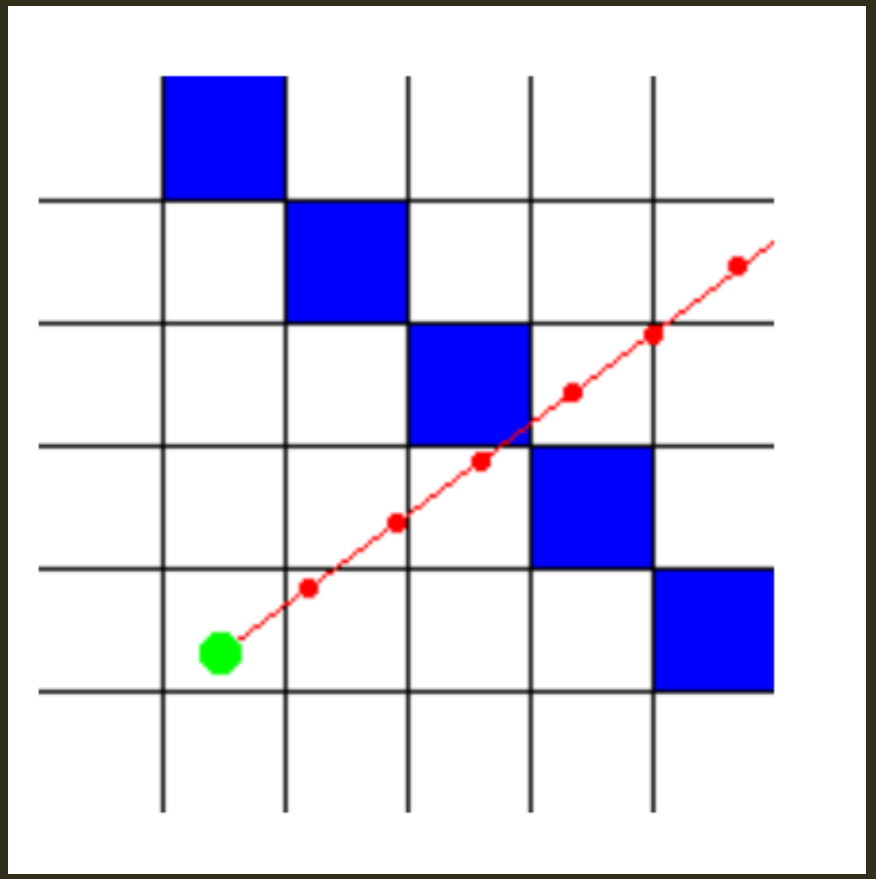
- Déplacement du joueur
- Ennemies en 2D
- Pouvoir tirer avec une arme
- Ennemies en 3D
- HUD
- Animations

LES DIFFICULTÉS

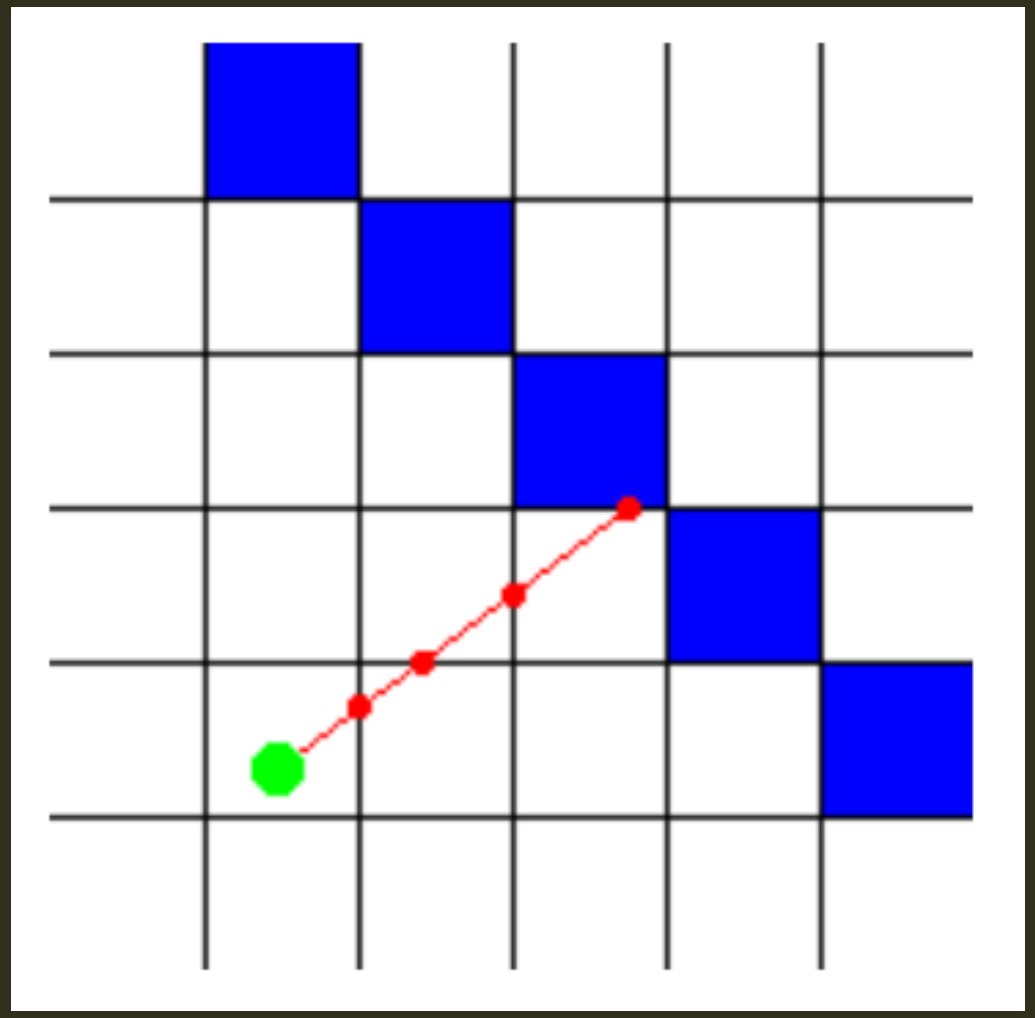
- Le rendu 3D et son optimisation qui nous a forcé a faire le passage de la librairie Bogue a OcamlSdl2
- Les collisions du joueurs
- La précision des tirs
- Fuite de mémoire: textures et affichage des textes

POURQUOI L'ALGORITHME DDA





POURQUOI CE CHOIX D'ALGORITHME



Algorithme DDA
+ Plus rapide
+ Plus fiable

dans un environnement quadrillé

L'ALGORITHME DE RAY CASTING DDA

```
deltaDist = { x = 1/vectRayDir.x; y = 1/vectRayDir.y }
pathVector = {x=0, y=0}
posInMap = {x=playerPos.x, y=playerPos.y}
step = { x = ±1, y = ±1 }
while( touchedWall(posInMap) ){
  if( pathVector.x < pathVector.y ){
    pathVector.x += deltaDist.x
    posInMap.x += step.x
  } else if( pathVector.x > pathVector.y ){
    pathVector.y += deltaDist.y
    posInMap.y += step.y
    side = 1
  }
}
```

TESTABILITÉ

- Nous avons uniquement effectué des tests manuel

CONCLUSION