COMPTE RENDU

Barjo Kart

Cailloux Alexandre, Bonbon Matthieu, Charamon Clément, Alvarez Maxence

L'objectif au travers de ce projet a été de concevoir des méthodes et des algorithmes permettant la création de trajectoires optimales sur des circuits.

1- Fonctionnalités implémentées

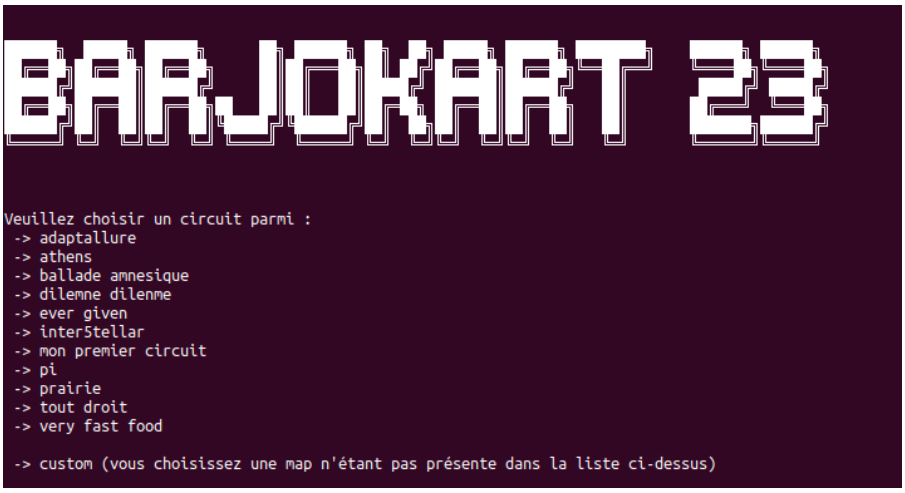
2- Algorithme de calcul de la trajectoire

3 - Répartition du travail

1- Fonctionnalités implémentées

Interface utilisateur (interface.cpp) :

- Sélection du circuit directement depuis la console
- Affichage d'informations utiles (simulation du score obtenu sur le circuit, informations concernant la génération du fichier binaire et de l'image des pixels visités



Algorithme de parcours du circuit « rustique » (parcours.cpp) :

- Parcours le circuit jusqu'à atteindre l'arrivée
- Utilisable principalement sur des petits circuits (temps de calcul très long sinon)

Autres fonctionnalités :

- Génération des circuits sous forme de tableau (circuit.cpp) :

```
Creation d'un tableau a 2 dimensions representant le circuit
9 = arrivee
0 = piste
2 = mur
```

(Format utilisé dans les algorithmes)

- Algorithme de génération d'un chemin sur un circuit à partir d'accélérations (circuit.cpp)

- Récupération des paramètres depuis un fichier TOML (recupParam.cpp)

2- Algorithme de calcul de la trajectoire

Afin d'implémenter un système de calcul de la trajectoire nous avons décidé de partir de l'algorithme de Dijkstra, permettant ainsi d'obtenir le plus court chemin.

1 - Initialisation

```
void Dijkstra::initDijkstra(){
  int nombrePointsMapDistance = 0; //Nombre de pixels à parcourir
  int nombreTotal = 0; //Nombre de pixels total sur l'image
  for(int i = 0; i < (int) (*this).circuit.size(); i++){
    for(int j = 0; j < (int) (*this).circuit[i].size(); j++){
      if(!(circuit[i][j]==2)){
        (*this).mapDistance[make_pair(i,j)]=INT_MAX;
        nombrePointsMapDistance++;
      }

      nombreTotal++;
    }
  }
  mapDistance[make_pair((*this).x_depart,(*this).y_depart)]=0;
}
```

Nous commençons par initialiser les distances dans la mapDistance.
Pour chaque pixel de l'image (sauf pour les murs, « if(!(circuit[i][j]==2)) ») on attribut la distance infinie (INT_MAX).
Le départ est initialisé à 0.

2 - Parcours des pixels du circuit

```
while (!PremiereArriveeTrouvee) {


  pair<int,int> m = (*this).minDist(); // pixel m

  compteurMinDist++;
  (*this).circuit[m.first][m.second] = 1;

  for(int i = m.first - 1; i <= m.first+1; i++ ){
    for(int j = m.second - 1; j <= m.second+1; j++ ){
      if(i >=0 && j >=0 && i < (int)(*this).circuit.size() && j < (int)(*this).circuit[m.first].size()){
        numberIteration++;
        maj_distances(m.first, m.second, i, j);
      }
    }
  }

}
vector<int> vitesses = chemin();
return vitesses;
```

Tant que nous ne trouvons pas de pixel représentant l'arrivée on continue de parcourir chaque pixel.
La fonction minDist() renvoie les coordonnées du pixel le plus proche du point de départ parmi les pixels non visités.

3 - MAJ des distances

```
void Dijkstra::maj_distances(int i, int j, int i1, int j1){

  bool voisin = false;
  if(i1==i+1 && j1==j && i != ((int)(*this).circuit.size() - 1)){
    voisin = true;
  }
  if(i1 == i-1 && j1==j && i!=0){
    voisin = true;
  }
  if(j1==j+1 && i1==i && j != ((int)(*this).circuit[i].size() - 1)){
    voisin = true;
  }
  if(j1 == j-1 && i1==i && j!=0){
    voisin = true;

  if(voisin && circuit[i1][j1] != 2) {
    if( ((*this).circuit[i1][j1]==0 || (*this).circuit[i1][j1]==9) && (*this).mapDistance.at(make_pair(i,j))!=INT_MAX &&
                                                (*this).mapDistance.at(make_pair(i,j))+1<(*this).mapDistance.at(make_pair(i1,j1)) ){

      (*this).mapDistance[make_pair(i1,j1)]=(*this).mapDistance[make_pair(i,j)]+1; //Ajout de 1 à la distance

      (*this).mapPredecessor[make_pair(i1,j1)]=make_pair(i,j); //Ajout du prédécesseur


      // on set l'arrivée
      if((*this).circuit[i1][j1]==9 && !PremiereArriveeTrouvee){

        (*this).arrivee=make_pair(i1,j1);
        cout << "Je suis tombé sur la fin" << endl;
        PremiereArriveeTrouvee=true;
      }
    }
  }
}
```

Pour chaque pixel, on vérifie que le pixel de coordonnées i1,j1 est bien un voisin valide (qu'il ne sorte pas de la taille du circuit).
Si le pixel est bien un voisin valide ET qu'il ne s'agit pas d'un mur, on vérifie que ce n'est pas un voisin déjà visité et que la distance est inférieur à la distance actuelle. Si tel est le cas, on l'ajoute an tant que prédécesseur.
Alors, si le pixel de coordonnées i1,j1 est l'arrivée (un « 9 » sur le circuit), on arrête le parcours du circuit : on a trouvé un chemin allant du départ à l'arrivée.

4 - Récupération du chemin

```cpp
vector<int> Dijkstra::chemin(){

  cout << "Affichage chemin" << endl;

  map<pair<int,int>,pair<int,int> > pred = (*this).mapPredecessor; // map des predecesseurs
  vector<pair<int,int> > listPoints; // liste de tt les points du chemin

  pair<int,int> Sdeb = make_pair((*this).x_depart,(*this).y_depart); // point de depart
  pair<int,int> currentSommet = (*this).arrivee;

  bool isValid = true;
  while(isValid){
    listPoints.insert(listPoints.begin(), currentSommet);
    currentSommet = mapPredecessor[make_pair(currentSommet.first, currentSommet.second)];
    if(currentSommet.first == Sdeb.first && currentSommet.second == Sdeb.second)
      isValid = false;
  }

  listPoints.insert(listPoints.begin(), Sdeb);
```

En partant du point d'arrivée et en passant uniquement pas les prédécesseurs, on atteint le point de départ : on a le chemin permettant d'aller du départ à l'arrivée pixel par pixel.

Sources : https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra
https://favtutor.com/blogs/dijkstras-algorithm-cpp

3- Répartition du travail

Maxence : Gestion des circuits (affichages, conversion image vers données manipulables, etc.), Parcours « rustique », morceau de Dijkstra, Optimisations de Dijkstra

Matthieu : Dijkstra, Récupérations paramètres TOML, Amélioration de l'interface (possibilité d'ajouter un nouveau circuit, affichage du score)

Alexandre : Interface, Génération des fichiers binaires à partir d'accélérations, Algorithme de lecture de trajectoire (en utilisant Bresenham)

Clément : Gestion des images, Génération des fichiers binaires à partir d'accélérations

Mode de fonctionnement du groupe : Majorité du travail effectué à l'université