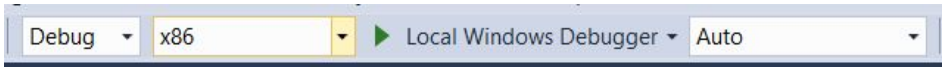
Tècniques d'Intel·ligència Artificial per a Videojocs

AA3: Pathfinding

Guia del codi

Compilació

Primer de tot recordeu que el projecte VS2017 només està configurat per a la plataforma x86. El projecte no compilarà per a x64 (a no ser que configureu vosaltres mateixos la solució per a aquesta plataforma).



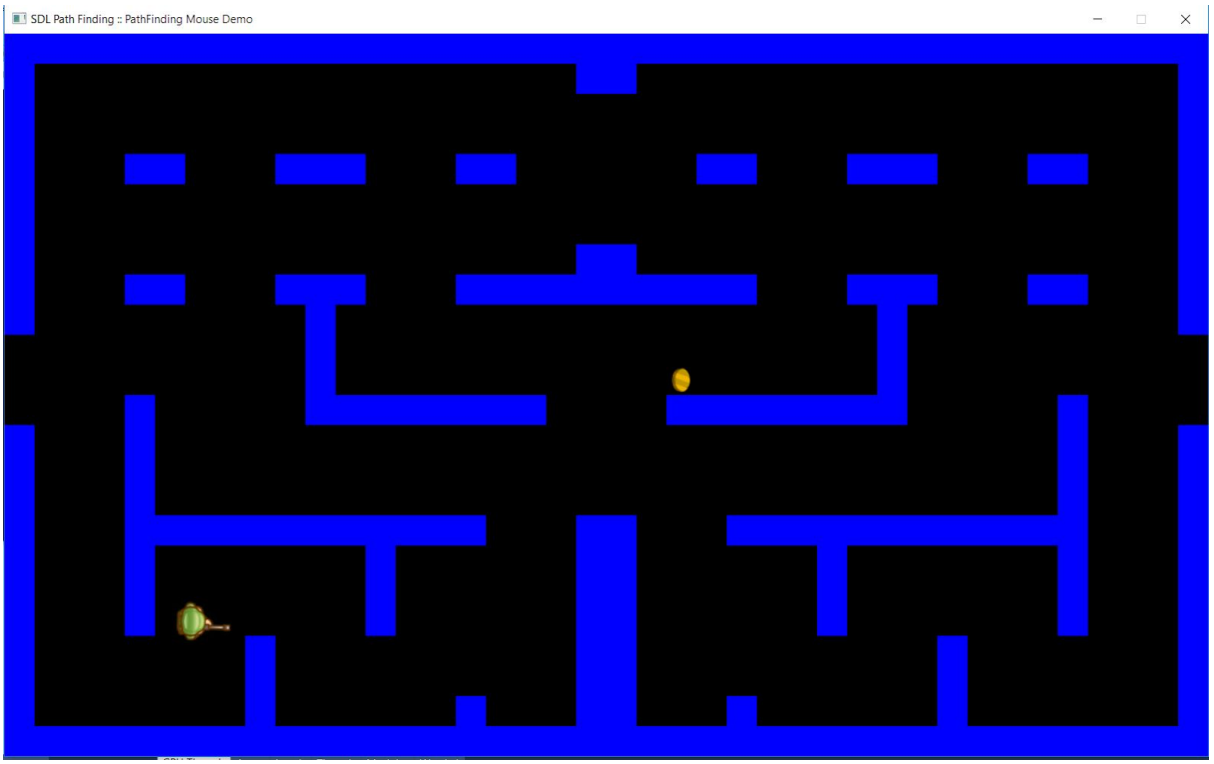
Estructura de classes

L'estructura bàsica del codi és la mateixa que heu fet servir per la pràctica AA1_Steering_Behaviors.

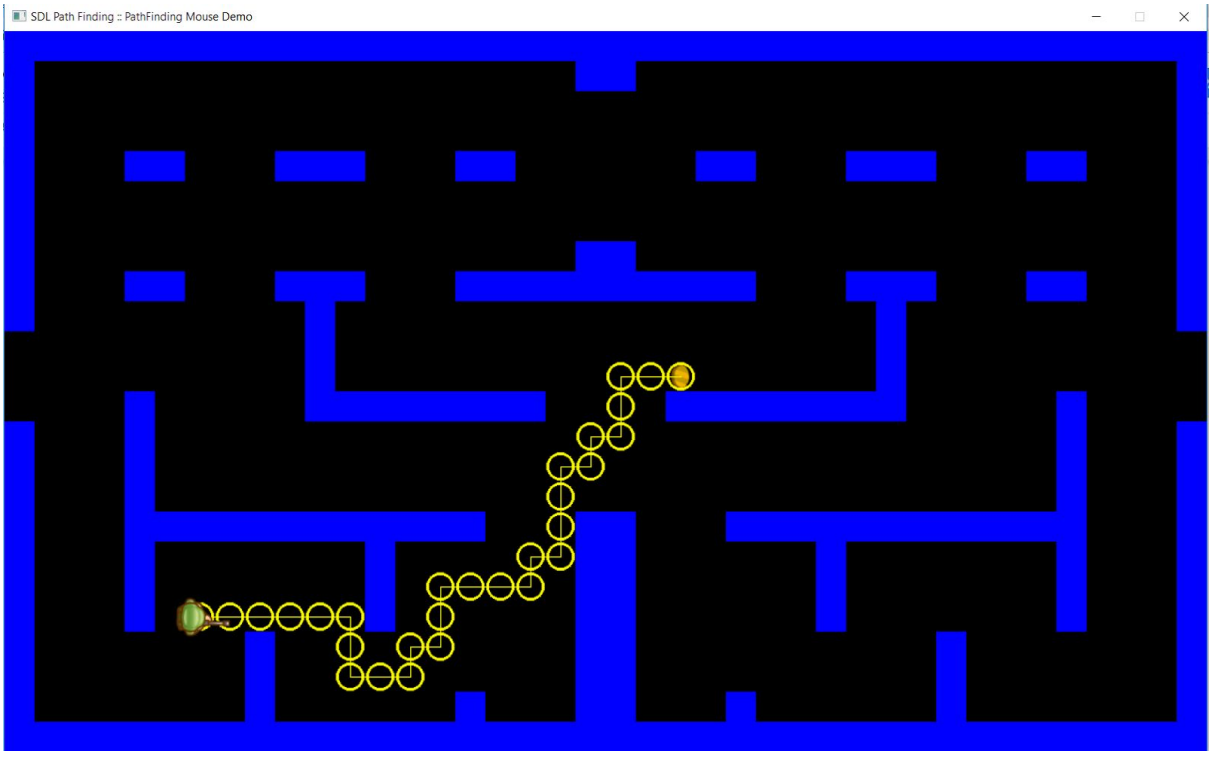
- **SDL_SimpleApp**: És la classe que crea i configura l'aplicació SDL (la finestra, el "renderer", etc.).
- **Scene**: Classe base per a crear escenes que s'executen en una instància de la classe *SDL_SimpleApp*. A cada frame, des de la funció `main()` estem cridant a la funció `SDL_Event SDL_SimpleApp::run(Scene *scene)` que pren com a paràmetre un punter a una escena i executa els seus mètodes `update()` i `draw()` (en aquest ordre).
- **Agent**: Implementa un agent intel·ligent i conté tota la informació de l'estat intern de l'agent. Normalment tindrem una o més instàncies de la classe ***Agent*** dins d'una escena, i anirem cridant els mètodes `update()` i `draw()` de cadascuna d'elles respectivament dins dels mètodes `update()` i `draw()` de l'escena.
- **SteeringBehavior**: Implementa els algorismes de Steering Behaviors. És una classe amiga de la classe **Agent**. Per aquesta pràctica se us proporciona una implementació dels tres comportaments que necessitareu: *Seek*, *Arrive*, i *SimplePathFollowing*.

L'escena ScenePathFindingMouse

Se us proporciona una escena de demostració per entendre el que s'ha de fer a la pràctica. En aquesta escena hi tenim un agent intel·ligent que s'ha de moure per arribar a una certa posició del nivell (marcada amb una moneda) tal com mostra la següent imatge:



El nostre agent haurà de trobar el camí òptim per a arribar fins a la posició de la moneda utilitzant algorismes de PAthFinding. En aquesta escena de demostració, en comptes de fer servir un algorisme de PathFinding, serem nosaltres qui marcarem el camí a seguir amb el ratolí. Premeu el botó esquerre del ratolí i moveu suaument el punter (sense deixar de prémer el botó) per a crear un camí.

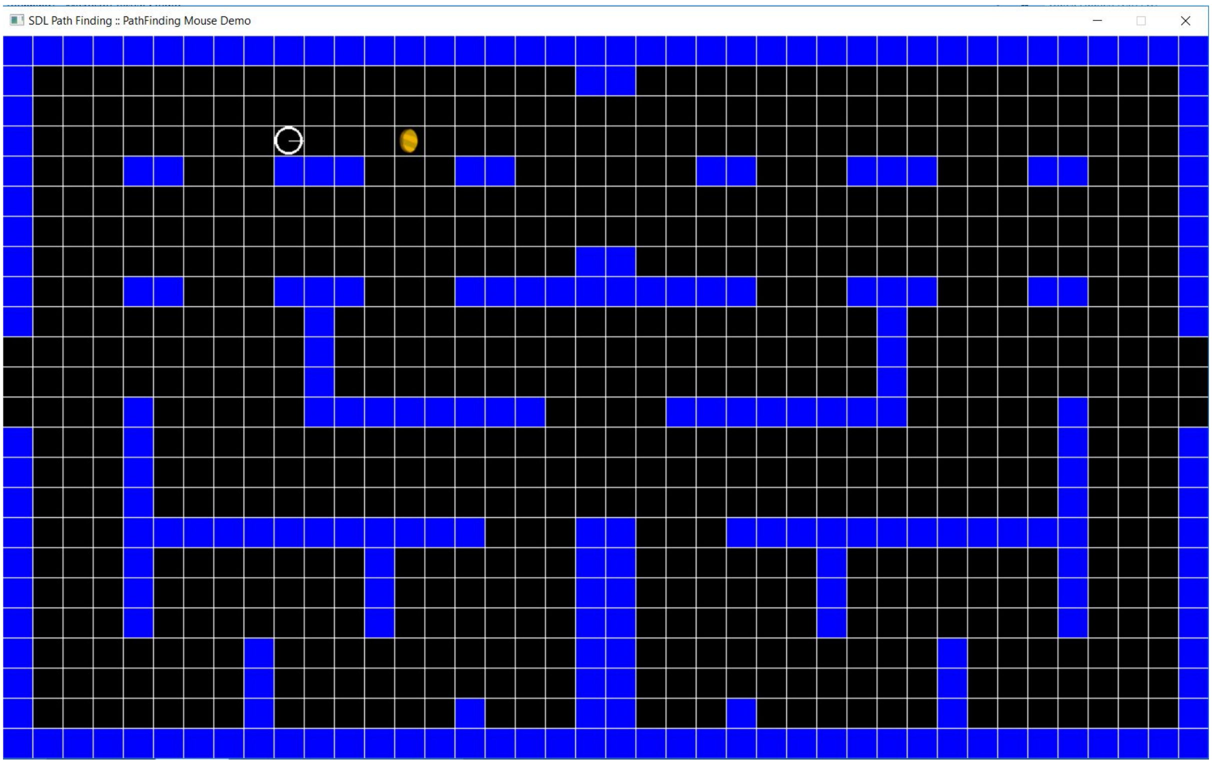


L'agent seguirà el camí marcat utilitzant l'algorisme SimplePathFollowing dels Steering Behaviors. Quan l'agent arriba a la posició de la moneda, aquesta és colocarà de nou en una posició aleatòria i l'agent haurà de tornar a buscar-la.

Path:
Estructura simple que ens permet representar camins (seqüències de punts) entre dues posicions del nivell.

```
struct Path
{
	std::vector<Vector2D> points;
	int ARRIVAL_DISTANCE = 35;
};
```

Representació Grid del nivell
Treballarem amb una graella ("grid") per a poder representar la geometria del nivell del joc. Per visualitzar la graella premeu la tecla "SPACE":



La geometria del nivell es crea dinàmicament a partir d'un fitxer de text. Per exemple, el nivell de la imatge anterior s'ha creat a partir del fitxer maze.csv que teniu a la carpeta SDL_PathFinding/res del projecte. El contingut d'aquest fitxer és el següent:

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0

```
0,1,1,1,0,0,1,1,1,0,0,0,1,1,1,0,0,1,1,1,1,1,1,0,0,1,1,1,0,0,0,1,1,1,0,0,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
0,1,1,1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,1,1,1,0,0,1,1,1,0
0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,0
1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1
1,1,1,1,0,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,1,1,1,1
0,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0
0,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0
0,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0
0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0
0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0
0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0
0,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,0
0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,0
0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,0
0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,0,0,1,1,1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Com podeu veure, en aquest fitxer representem els murs amb el valor “0” i l’espai obert (per on es mourà l’agent) amb el valor “1”. Podrem representar diferents tipus de terreny utilitzant altres valors (>=1) si cal.

Tota la informació de la geometria del nivell es guarda en una variable de l’escena:

```
std::vector< std::vector<int> > terrain;
```

La funció **`initMaze`** llegeix l’especificació del nivell des d’un fitxer i la desa a la variable **`terrain`**:

```
void ScenePathFindingMouse::initMaze(char* filename) {}
```

IMPORTANT: Aquesta manera de representar el nivell (grid) implica que treballarem amb dos sistemes de coordenades diferents: (1) la matriu de píxels (la finestra de SDL_SimpleApp); i (2) la matriu de cel·les (**`terrain`**). En alguns casos ens caldrà convertir entre coordenades de cel·la i coordenades de píxel. Per exemple, si el volem saber a quina cel·la es troba el nostre agent haurem de convertir les coordenades de píxel (**`Vector2D(253, 468)`**) que tenim a **`agent->position`** a coordenades de cel·la, i d’altre banda si volem que l’agent es dirigeixi a un cel·la en concret (p.ex. **`terrain[1][1]`**) haurem de calcular-ne les coordenades a nivell de píxel per a poder fer un Seek fins allà. Teniu dues funcions ja implementades per a fer aquestes conversions:

```
Vector2D ScenePathFindingMouse::cell2pix(Vector2D cell) { }
Vector2D ScenePathFindingMouse::pix2cell(Vector2D pix) { }
```

Classes que s'han d'implementar

Haureu d'implementar (com a mínim) les següents classes:

- **Graph**: Classe que implementa un graf. Haurà de tenir un constructor que prengui com a paràmetre una variable de tipus `std::vector< std::vector<int> > *terrain`, en la que tenim la informació del "grid" de l'escena (veure escena PathFindingMouse).
- **Node**: Implementa un node d'un graf.
- **Edge**: Implementa una aresta d'un graf.
- **PathFinding**: Classe que implementa els diferents algorismes de Path Finding. Una instància de la classe Agent haurà de tenir accés a aquesta classe i cridarà els seus mètodes per a trobar camins entre dos nodes d'un graf, per exemple:

  ```
  Path PathFinding::DFS(Graph* graph, Node* start, Node* goal)
  Path PathFinding::Dijkstra(Graph* graph, Node* start, Node* goal)
  ```

  Tot el codi dels algorismes de PathFinding ha d'estar dins d'aquesta classe.