> Othello - Reversi

Joc de tauler 2D en Lua

L'Othello – Reversi és un joc de taula d'estratègia per dos jugadors que es juga en un tauler de 8x8. Els jugadors disposen d'un total de 64 peces circulars, que tenen dues cares, una negra i una blanca. Durant la partida, els jugadors intenten situar les peces sobre el tauler de forma que les peces del jugador contrari quedin «atrapades» entre dues peces pròpies, seguint una línia recta (horitzontal, vertical o diagonal). Les peces «atrapades» canvien de color i passen a ser del jugador que ha efectuat el movment. El joc finalitza quan ja s'han esgotat totes les 64 peces , o bé no es pot realitzar cap moviment vàlid. Guanya el jugador que té més fitxes del seu color. En aquesta pràctica, desenvluparem el joc reversi en 2D mitjançant el motor LÖVE2D per Lua.

Data d'entrega:
28/11/2016 23:59 GMT+1

Presentació al professor (durant la classe de pràctiques):
A concretar

1. Objectiu

L'objectiu d'aquesta pràctica és que l'alumne practiqui amb les funcionalitats bàsiques del motor LÖVE 2D, per desenvolupar jocs en Lua, i apliqui les tècniques de desenvolupament habituals de dinàmiques senzilles de joc (bucle de joc).

S'utilitzarà el llenguatge de programació interpretat Lua un entorn de desenvolupament com ZeroBrane, que permet la integració del motor LÖVE 2D i el desenvolupament i execució directa de jocs en Lua.

2. Resum general

La pràctica consisteix en el desenvolupament en Lua del joc Othello – Reversi, mitjançant el motor LÖVE 2D i les seves funcionalitats bàsiques de gràfics, audio i maneig de teclat i ratolí. L'objectiu de la pràctica es centra en la programació de la dinàmica de joc, per la qual cosa els gràfics seran senzills, en principi sense la utilització de *sprites* animats. Les funcions de dibuix del LÖVE seren suficients per dissenyar l'entorn de joc.

En tractar-se d'un joc de tauler clàssic de dos jugadors, es poden donar dues situacions: que els contrincants siguin dos jugadors humans o bé que un d'ells sigui el propi dispositiu (IA). La implementació bàsica és la d'un joc amb dos jugadors humans, l'altra és opcional. No obstant, cal tenir en compte que si es vol desenvolupar aquesta opció, no cal entrar en algorismes d'intel·ligència artificial per trobar la millor jugada per part de la màquina. Serà suficient que esculli la primera jugada possible disponible o bé que ho faci de forma aleatòria.

3. Consideracions prèvies i entrega

La pràctica es desenvoluparà individualment.

El codi ha de ser clar i entenedor. És recomanable utilitzar comentaris per explicitar el funcionament de determinades parts del codi, sobretot aquelles en què la funcionalitat no s'apreciï a simple vista.

Cal assegurar-se d'haver eliminat codi de prova si al final no s'executa i no forma part del programa

Cada alumnes entregarà el fitxer o fitxers de codi font (.lua), a través del campus virtual. **Cada fitxer de codi font inclourà explícitament (com a comentari) el nom dels autors de la pràctica**

La data límit d'entrega és l'assenyalada a l'inici de l'enunciat.

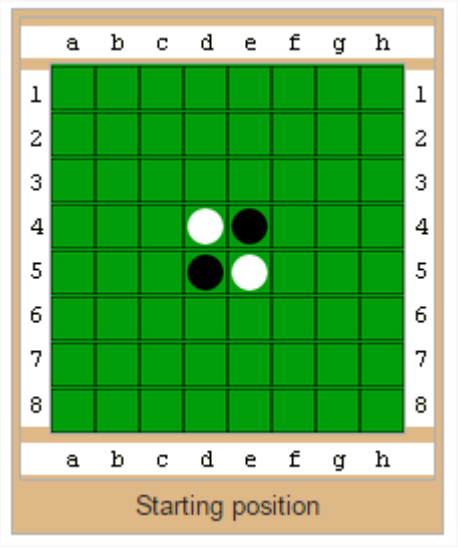
A banda de l'entrega dels fitxers del codi font, caldrà fer una presentació de la pràctica al professor (5-10 minuts), durant les classes indicades a l'inici de l'enunciat. En aquesta presentació caldrà demostrar la funcionalitat del programa (encara que no estigui acabat), exposar la solució desenvolupada, les característiques principals del programa, les dificultats sorgides i la manera de resoldre-les. El professor podrà realitzar preguntes i comprovar el funcionament del programa. Aquesta presentació **forma part de l'avaluació de la pràctica**, així com de la nota d'assistència, participació i activitats complementàries, i **és individual** (pot ser diferent per a dos membres del mateix grup).

La dificultat de la pràctica és creixent. Es valorarà tant el grau de funcionalitat aconseguida com la compactació, claredat i eficiència del codi desenvolupat.

4. Regles del joc

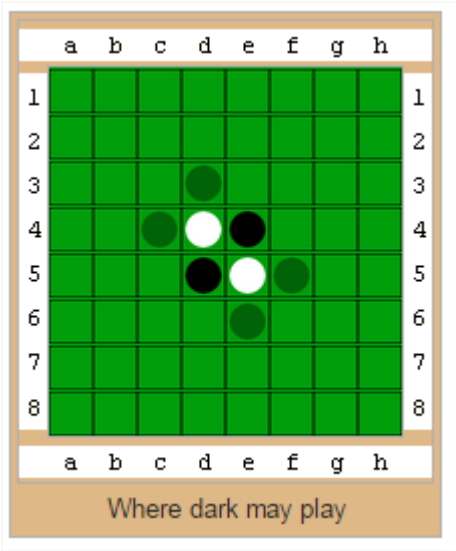
- Es disposa de 64 fitxes en total, pels dos jugadors. Les fitxes tenen dues cares (una blanca i una negra, «light», «dark»)

-La posició inicial del joc és la següent:



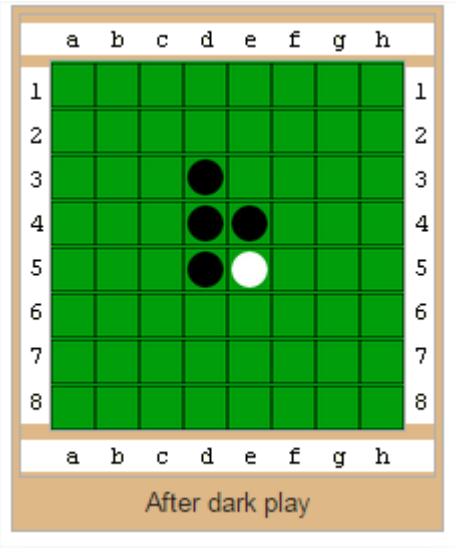
- Les negres fan el primer moviment, que consisteix en situar una nova fitxa sobre el tauler. Després el torn passa a l'altre jugador, i així successivament. Els moviments sempre consisteixen en situar noves fitxes sobre el tauler. Les fitxes que ja s'han col·locat no es poden moure (excepte canviar-les do color, com veurem més endavant).

- Només és possible situar una fitxa en una posició si amb el moviment «s'atrapen» fitxes del contrari. Les fitxes d'un color queden «atrapades» quan hi ha una línia recta de fitxes (horitzontal, vertical o diagonal) entre dues fitxes de l'altre color. Per exemple, el cas anterior els moviments vàlids serien els següents:

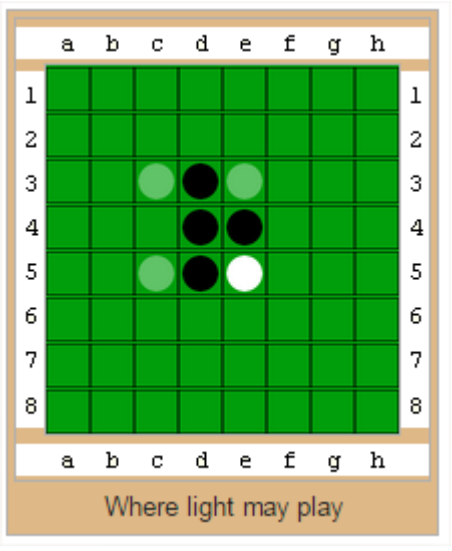


- Un moviment que no permeti capturar fitxes del contrari no es considera un moviment vàlid. Si un jugador no pot efectuar cap moviment vàlid, passa el torn al contrari.

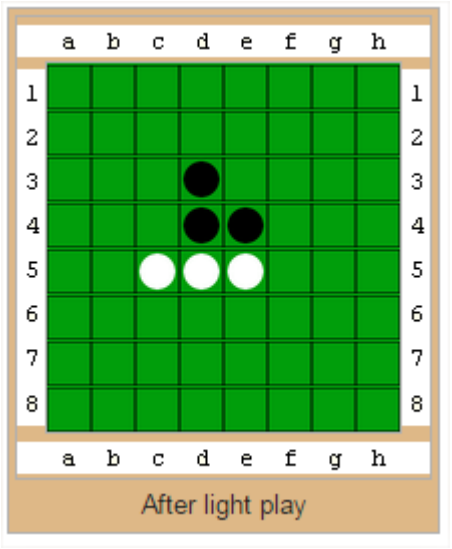
Un cop efectutat el moviment, totes les fitxes «atrapades» del contrari, canvien de color. Per exemple, si en el cas anterior la fitxa negra s'hagués situat a la posició d3, un cop girades les fitxes corresponents el tauler quedaria així:



- A continuació, és el torn de les blanques. Els moviments possibles són els següents:



Suposant que el jugador situi la fitxa blanca a c5, un cop girades les fitxes corresponents el tauler quedaria així:



El joc finalitza quan cap jugador pot realitzar un moviment vàlid, o quan s'han esgotat totes les fitxes.

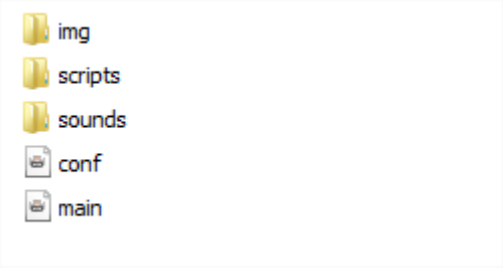
Guanya la partida el jugador que té més fitxes del seu color situades sobre el tauler.

Més informació sobre el joc:
https://en.wikipedia.org/wiki/Reversi

5. Estructura del projecte

Per desenvolupar el joc, seguirem un disseny funcional descendent, dividint la tasca en diverses funcions.

En primer lloc, l'estructura de la carpeta de projecte serà similar a aquesta:



- El fitxer script **conf.lua**, té la configuració inicial de la finestra de joc

```
function love.conf(t)
...
end
```

- El fitxer script **main.lua** té les funcions del bucle de joc:

```
function love.load()
...
end

function love.update(dt)
...
end

function love.draw()
...
end
```

A la carpeta **scripts** podeu situar els fitxers de script auxiliars, on posar determinades funcions i codi si voleu evitar que el fitxer **main.lua** sigui massa llarg. Per exemple:

- drawingFunctions.lua
- gameFunctions.lua
- etc...

A la carpeta **sounds** podeu situar els efectes de so del joc.
A la carpeta **img** podeu situar les imatges utilitzades en el joc (si s'utilitzen com a sprites, HUD, etc.)

L'estructura anterior és orientativa. Podeu fer la vostra, tenint en compte que la carpeta de projecte ha de tenir una estructura **clara i entenedora**, i que sempre ha d'incloure els fitxers **main.lua** i **conf.lua** a la **carpeta arrel**.

Treballeu amb la consola activada per poder debugar el codi més fàcilment.

6. Mecànica de joc

El joc bàsic serà amb dos jugadors (humans), que disposaran de tecles diferents per jugar (d) o (l), per exemple. També podeu implementar jugabilitat amb els dos botons del ratolí. Caldrà realitzar un control de torns del jugador corresponent.

6. Guia de desenvolupament (primer capítol...)

Per desenvolupar el joc, podeu seguir la següent guia de desenvolupament, on s'indiquen les variables principals, les funcions a programar, els paràmetres d'entrada i valor a retornar.

```
local gameBoardBoxes = {
{x = nil, y = nil diskType = nil},
{x = nil, y = nil diskType = nil},
..
{x = nil, y = nil diskType = nil},
```

Aquesta variable és la taula principal del joc. Conté 64 elements (un per cada casella del tauler), amb les coordenades x, y corresponents al centre de la casella. Cada element també té un camp diskType que conté el tipus de fitxa que ocupa aquella casella:
- "dark": fitxa negra
- "light": fitxa blanca
- nil: no hi ha fitxa

Recordeu que no cal definir-li uns valors inicials, només declarar-la al principi

```
local gameBoardBoxes = {}
```

function initGameBoardBoxes(numRows, numColumns)

Aquesta funció retorna una taula amb «numRows» x «numColums» elements, seguint l'estructura de gameBoardBoxes i els valors inicials corresponents: la x i la y corresponents al centre de cada casella, i diskType inicialitzada a nil.
Per exemple, si volguéssim inicialitzar un tauler de 3 x 3 en una finestra de 600 x 600 px:

```
gameBoardBoxes = initGameBoardBoxes(3, 3)
..
--gameBoardBoxes:
-- {100, 100, diskType = nil},
-- {300, 100, diskType = nil},
..
– {500, 500, diskType = nil}
```

function drawGameBoardGrid(numRows, numColumns)

Aquesta funció dibuixa les línies separadores del tauler de joc, de numRows x numColumns caselles. Utilitzeu mides relatives a la mida de la finestra principal.
Recordeu que el tauler és una estructura de 8x8, però cal utilitzar els paràmetres numRows i numColumns que es passen a la funció. Per dibuixar les línies podeu utilitzar la senzilla funció «Line» del motor LÖVE

La funció no retorna cap valor.

function drawGameBoardBoxes()

Aquesta funció dibuixa fitxes del tauler de joc. Recordeu que la situació global del tauler està guardada a la variable gameBoardBoxes (passada com a argument).
Per dibuixar les fitxes podeu utilitzar la senzilla funció «Circle» del motor LÖVE.
La funció no retorna cap valor.

function getSelectedGameBoardBox(x, y)

La funció determina la casella seleccionada (via teclat o ratolí) pel jugador per posar-hi una fitxa.
La funció retorna la fila i la columna de la casella seleccionada.

Exemple en taule 3x3 i finestra de 600 x 600:

```
box = getSelectedGameBox(100, 150)
..
-- box:
-- (1, 1)
```

function addDiskToGameBoard(row, column, diskType)

La funció afegeix una fitxa de tipus «diskType» a la taula gameBoardBox, a la casella [row][column]. Dins la funció cal haver comprovat que la casella està buida.

En cas que la casella estigui buida, retorna true. En cas que estigui ocupada, retorna false.

IMPORTANT: NO cal comprovar que es tracti d'un moviment vàlid (atrapant fitxes), ho farem posteriorment.

Exemple: afegim una fitxa «light» a la posició [1][1] del tauler:

```
--gameBoardBoxes:
-- {100, 100, diskType = nil},
-- {300, 100, diskType = nil},
..
– {500, 500, diskType = nil}

addDiskToGameBoard(1, 1, "light")

-- true
--gameBoardBoxes:
-- {100, 100, diskType = "light"},
-- {300, 100, diskType = nil},
..
– {500, 500, diskType = nil}
```

Aquesta guia de funcions s'anirà ampliant en propers dies...

7. Consells i pistes

- Dividiu les tasques del programa en múltiples funcions, que tinguin assignades tasques clares, amb paràmetres clars i retorns coherents.
- Algunes funcions hauran de realitzar tasques molt bàsiques, com ara multiplicar matrius. Altres realitzaran tasques més complexes cridant a aquestes. No dubteu en simplificar les funcions i subdividir la feina tant com calgui.
- Utilitzeu la potència de Lua en el tractament de taules
- Utilitzeu l'estructura de *for* genèric allà on sigui possible
- Utilitzeu els retorns múltiples de funcions, una característica especial de Lua
- Realitzeu control d'errors, recordeu que el tipat dinàmic de dades pot donar resultats inesperats.
- Teniu l'operador matemàtic % per el residu de la divisió entera.

- No existeix una solució única, la programació no és una "ciència" exacta. Les solucions aportades seran totes avaluades per la seva funcionalitat, eficiència i claredat del codi.