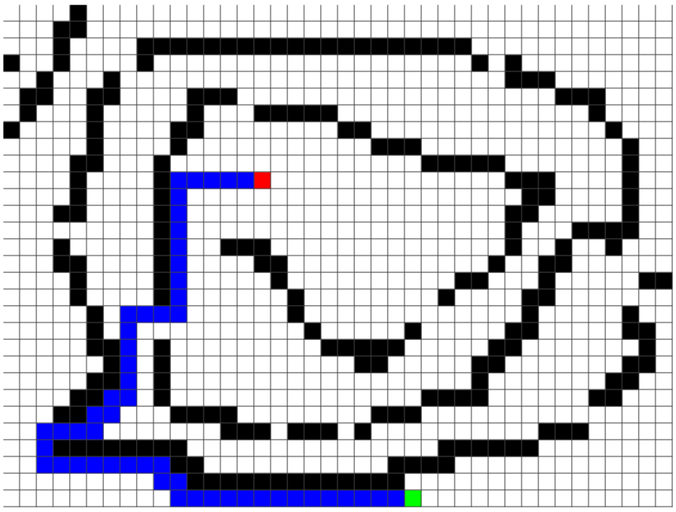

ADSG - Opdracht 5

Dijkstra

Simon Karman - 500621839
Jorn Theunissen - 500621813

22-11-2011

Inhoud

Met behulp van het Dijkstra Algoritme gaan we een leuke pathfinding demo maken. Om het onszelf makelijk te maken beginnen we direct met het maken van een Tile class.

Tile Class

```
public class Tile
{
        //Variables
        public var tileType:String;
        public var textfield:TextField;
        public var added:Boolean;
        public var shortest:Number;
        //if added -> shortest staat vast
        //if not added -> shortest staat nog niet vast

        //Constructor
        public function Tile(main:Main, tileType:String, x:Number, y:Number) {
                this.tileType = tileType;
                this.added = false;
                this.shortest = Number.MAX_VALUE;
                //Textfield Shit
                this.textfield = new TextField();
                this.textfield.text = "";
                this.textfield.x = x;
                this.textfield.y = y;
                //main.addChild(textfield);
        }
}
```

De belangrijkste variabelen zijn ‘Added’ en Shortest, Added houd bij of de tile aan de buitenste rand van gecontroleerde tiles ligt. Iederee keer wanneer het algoritme wordt aangeroepen beginnen we met het checken of de huidige tile aan een randtile ligt. De Shortest tile houd daarnaast bij hoeveel stappen er nodig zijn om bij deze tile te komen vanaf de start.

Main class

```
var arr:Array;

public function Main():void
{
        if (stage) init();
        else addEventListener(Event.ADDED_TO_STAGE, init);
}

private function init(e:Event = null):void
{
        removeEventListener(Event.ADDED_TO_STAGE, init);
        // entry point

        arr = BitmapToArray.toArray(this, test1PNG);

        var dijkstra:Dijkstra = new Dijkstra();
        dijkstra.getAdjacentieMatrix(arr);

        dijkstra.drawShortestPath(arr);

        BitmapToArray.drawArray(this, arr, graphics, 20);

        stage.addEventListener(MouseEvent.CLICK, mouseClicked);
}
```

De main class bevat de array die de grid bij houd en een Dijkstra object. Het dijkstra object gebruikt de methode ‘GetAdjacentieMatrix’ om dijkstra toe te passen en te kijken wat de loop afstanden zijn.

Dijkstra class

De dijkstra class is de grootste (en leukste) class die in het programma zit.

GetAdjacentieMatrix

```
public function getAdjacentieMatrix(grid:Array):void {
        //Find start and end position
        for (var i:int = 0; i < grid.length; i++)
        {
                if (grid[i].tileType == TileType.START)
                {
                        (grid[i] as Tile).added = true;
                        (grid[i] as Tile).shortest = 0;
                        (grid[i] as Tile).textfield.text = (grid[i] as Tile).shortest.toString();
                }
                if (grid[i].tileType == TileType.END)
                {
                        (grid[i] as Tile).textfield.text = ":)";
                }
                if (grid[i].tileType == TileType.PATH)
                {
                        (grid[i] as Tile).tileType = TileType.WALKABLE;
                }
        }

        var gridWidth:Number = 40;
        var gridHeight:Number = 30;

        //vind aangrenzende tiles
        var adjacentTiles:Array = new Array();
        var notEnd:Boolean = true;

        //while loop
        while (notEnd)
        //for (var jj:int = 0; jj < 20; jj++)
        {
```

Deze functie krijgt de array mee die alle tiles bevat. De start tile wordt als ‘added’ neergezet en krijgt de waarde true. Dit is belangrijk omdat we willen gaan kijken of een tile zich naast een adjacentTile bevind. De eindtile krijgt een andere text om duidelijk te maken dat deze het einde is. Alle tiles die van het type path zijn worden overigens ook op walkable gezet. Dit wordt gedaan om ze opnieuw mee te nemen in de beloopbare grid. We gebruiken een while loop die alle tiles gaat doorlopen. In de AdjacentTiles array houden we bij welke tiles als laatste gechecked zijn en nog nodes naast zich hebben liggen.

```
//for (var jj:int = 0; jj < 20; jj++)
{
	//loop door tiles op zoek naar aangrenzende tiles en houd de gevonden kleinste kortste-afstand bij
	adjacentTiles.length = 0;
	var smallestShortestFound:Number = Number.MAX_VALUE;
	for (var j:int = 0; j < grid.length; j++)
	{
		var tile:Tile = grid[j];
		//(is niet niet added) en GEEN nietbeloopbare tile
		if (!tile.added && !(tile.tileType == TileType.NONWALKABLE)) {
			var index:Number = 0;
			var found:Boolean = false;
			//is een aangrenzende added? -> aangrenzend
			//Boven
			index = j - gridWidth;
			if ((index >= 0) && (grid[index] as Tile).added) {
				//trace("Boven");
				tile.shortest = (grid[index] as Tile).shortest + 1;
				tile.textfield.text = tile.shortest.toString();
				found = true;
			}
			//Onder
			index = j + gridWidth;
			if ((index < (gridWidth * gridHeight)) && (grid[index] as Tile).added) {
				//trace("Onder");
				tile.shortest = (grid[index] as Tile).shortest + 1;
				tile.textfield.text = tile.shortest.toString();
				found = true;
			}
			//Rechts
			index = j + 1;
			if ((!((index % gridWidth) == 0)) && (grid[index] as Tile).added) {
				//trace("Rechts");
				tile.shortest = (grid[index] as Tile).shortest + 1;
				tile.textfield.text = tile.shortest.toString();
				found = true;
			}
			//Links
			index = j - 1;
			if ((!((j % gridWidth) == 0)) && (grid[index] as Tile).added) {
				//trace("Links");
				tile.shortest = (grid[index] as Tile).shortest + 1;
				tile.textfield.text = tile.shortest.toString();
				found = true;
			}
```

Het grootste gedeelte van de while loop zit hem in het doorlopen van elke tile. Voor elke tile die Niet non-walkable is gaan we boven, rechts, onder en links checken. We houden een variabele bij om makkelijk te bepalen wat de naastgelegen tile voor index heeft. Wanneer er een tile naast de huidige tile ligt en deze heeft de boolean waarde ‘Added’ of true, dan betekend dat dat we een nieuwe shortest waarde moeten bereken. Dit is simelweg het shortest nummer van de onderzochte tile nemen en hier 1 bij op tellen. We zetten tevens de boolean found op true.

```
if (found) {
	if (smallestShortestFound > tile.shortest) {
		smallestShortestFound = tile.shortest;
		adjacentTiles.push(tile);
	}
}
```

When the boolean is found is to true, we are going to check if the smallesShortestFound is higher than the tile.shortest we are current investigating. We set the value of the tile.shortest as the new smallesShortestFound. We also add this tile to the adjacentTiles array.

```
var addedAdjacentTile:Boolean = false;
for (var k:int = 0; k < adjacentTiles.length; k++)
{
        var adjacentTile:Tile = adjacentTiles[k];
        if ((adjacentTile.shortest == smallestShortestFound) && (!addedAdjacentTile)) {
                adjacentTile.added = true;
                adjacentTile.shortest = smallestShortestFound;
                adjacentTile.textfield.text = smallestShortestFound.toString();
                addedAdjacentTile = true;
                if (adjacentTile.tileType == TileType.END) {
                        notEnd = false;
                }
        } else {
                adjacentTile.textfield.text = "";
                adjacentTile.added = false;
                adjacentTile.shortest = Number.MAX_VALUE;
        }
}
if (!addedAdjacentTile) {
        notEnd = false;
}
```

Nu gaan we kijken wat per aangrenzende tile het korste pad is. Als het korste padgevonden gelijk is aan het bekendste korste pad en de tile is nog niet toegevoegd, dan zetten we deze tile adjacentTile op 'added' true. We kijken tevens ook of de beken tile niet de end tile is. Wanneer dit zo zou zijn, dan kunnen we stoppen met de search. Wanneer de tile waarde groter is dan het korste pad dan, is de tile niet meer added.

DrawShortestPath

```
public function drawShortestPath(grid:Array):void {
        var gridWidth:Number = 40;
        var gridHeight:Number = 30;

        var currentTile:Number;
        var back:Boolean = false;

        for (var i:int = 0; i < grid.length; i++)
        {
                var tile:Tile = grid[i];
                if (tile.tileType == TileType.END) {
                        currentTile = i;
                }
        }

        while (!back) {
                var index:Number = 0;
                var shortest:Number = Number.MAX_VALUE;
                var nextTile:Number = -1;
                //Boven
                index = currentTile - gridWidth;
                if ((index >= 0) && (grid[index] as Tile).added) {
                        if ((grid[index] as Tile).shortest < shortest) {
                                nextTile = index;
                                shortest = (grid[index] as Tile).shortest;
                        }
                }
```

We houden bij wat de huidige tile is. Vanaf daar kunnen gaan zoeken naar een waarde die net 1 waarde lager zit dan de huidige tile. Wanneer deze statement true is, wordt die tile de volgende tile waar we van uit gaan zoeken. De tile wordt tevens ook ingekleurd.

Extra

Het zat niet in de opdracht maar we hebben ook wat toegevoegd zodat je de eind tile kan bepalen door de tile te selecteren met de muis.

```
public function mouseClicked(e:MouseEvent):void {
	var x:Number = Math.floor(mouseX / 20);
	var y:Number = Math.floor(mouseY / 20);
	if (((arr[x + y * 40] as Tile).tileType == TileType.WALKABLE) || ((arr[x + y * 40] as Tile).tileType == TileType.PATH)) {
		for (var i:int = 0; i < arr.length; i++)
		{
			var item:Tile = arr[i];
			item.added = false;
			item.shortest = Number.MAX_VALUE;
			item.textfield.text = "";
			if (item.tileType == TileType.END) {
				item.tileType = TileType.WALKABLE;
			}
		}

		(arr[x + y * 40] as Tile).tileType = TileType.END;

		var dijkstra:Dijkstra = new Dijkstra();
		dijkstra.getAdjacentieMatrix(arr);
		dijkstra.drawShortestPath(arr);

		BitmapToArray.drawArray(this, arr, graphics, 20);
	}
}
```