Nibbles

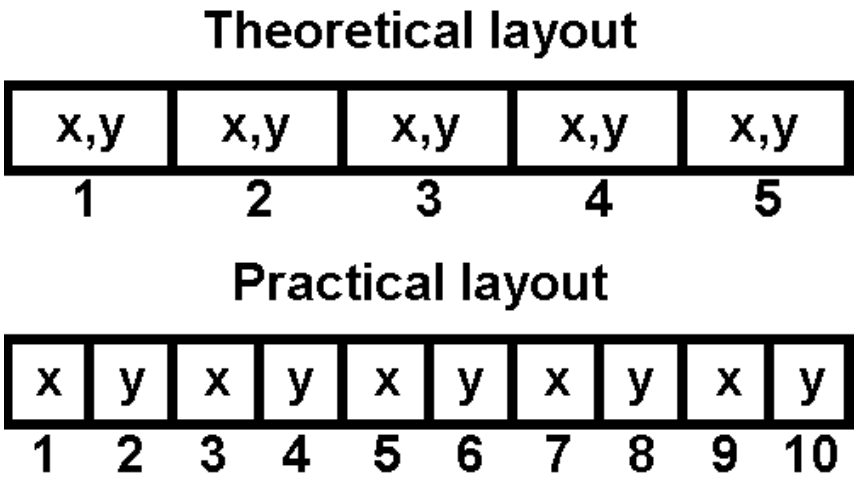
Jonas Andersson and Mattias Lunderot
DV1463 - Performance Optimization

Implementation

We have implemented the game Nibbles in Assembly using AT&T syntax. Nibbles is a game where you play as a snake who eats apples. The apples are randomly placed on the field, and when the snake eats an apple it grows. If the snake collides with itself or the borders of the field it dies.

We started implementing the game in C (The code can be seen in the Appendix-section), and then we translated it as close as possible to Assembly code. In C we use a struct called Node, which holds the X and Y value of the position. The snake is then made up of an array of Nodes, where each node is a position of a body part. The same goes for the apples, where each node is a position of an apple.

In theory we chose to implement it using the same approach in Assembly. As assembly doesn't support that advanced arrays, the practical implementation looks as follows:



To get the first part of the body we have to get the first and second element, to get the second part of the body we get third and fourth element, and so on.

Assembly-related

The only major difference between our C-implementation and our Assembly-implementation is the above mentioned difference in the way to handle the array. Other than that it's just that Assembly needs more code for "simple" tasks like for example looping through the array than C does.

Otherwise it was a pretty straight forward conversion between the two.

Optimisation

We tried to keep the number of loops as few as possible. For example when drawing the borders to the screen we only use one loop for all four edges, instead of one loop per edge.

We also minimized the memory used by using separate arrays for the apples and the snake instead of one single two-dimensional array for the whole field.

Could have used LEAL instruction to, for example, get the offset of the address to the body parts, which would have shortened those parts significantly.

We also could have created functions for often duplicated code, like getting the offset in the array.

Appendix

```
typedef struct
{
	int x, y;
} Node;

void start_game(int len, int n_apples)
{
   nib_init();

   /*Constants*/
   const int screenSize = 50;
	const int maxLen = 50;
	const int maxApples = 100;
	const int sleepTime = 10000;

	int currentLength = len;
	int direction = 0;

	int hit = 0;
	int done = 0;
	int input = -1;

	Node body[maxLen];
	Node apples[maxApples];

	/*Init snake body*/
	int i;
	for (i = 0; i < len; i++)
	{
		body[i].y = screenSize/2;
		body[i].x = screenSize/2 - i;

	}

	/*Init apples*/
	for (i = 0; i < n_apples; i++)
	{
		apples[i].x = rand() % screenSize;
		apples[i].y = rand() % screenSize;
	}









	while(!done)
	{
```

```
/*Update*/
Node pos = body[0];
if (direction == 0)
{
	body[0].x++;
}
else if (direction == 1)
{
	body[0].y--;
}
else if (direction == 2)
{
	body[0].x--;
}
else if (direction == 3)
{
	body[0].y++;
}

/*Collision detection with apples*/
hit = 0;
for (i = 0; i < maxApples; i++)
{
	if (apples[i].x == body[0].x &&
		apples[i].y == body[0].y )
	{
		hit = 1;
		apples[i].x = rand() % screenSize;
		apples[i].y = rand() % screenSize;
		break;
	}
}
if (hit)
{
	body[currentLength] = body[currentLength-1];
}
for (i = 1; i < currentLength; i++)
{
	//xor swapping
	Node temp;
	temp = body[i];
	body[i] = pos;
	pos = temp;
}
if (hit)
{
	currentLength++;
}

/*Collision detection with self*/
for (i = 1; i < currentLength; i++)
{
	if (body[i].x == body[0].x &&
		body[i].y == body[0].y )
	{
		done = 1;
		break;
	}
}

/*Collision detection with walls*/
for (i = 0; i < screenSize; i++)
{
```

```
            if ((body[0].x == i && (body[0].y == 0 || body[0].y == screenSize)) ||
                (body[0].y == i && (body[0].x == 0 || body[0].x == screenSize))
                )
            {
                done = 1;
                break;
            }
        }

        /*Draw snake*/
        for (i = 0; i < currentLength; i++)
        {
            nib_put_scr(body[i].x, body[i].y, 'O');
        }

        /*Draw apples*/
        for (i = 0; i < maxApples; i++)
        {
            nib_put_scr(apples[i].x, apples[i].y, '*');
        }
        /*Draw game borders*/
        for (i = 0; i < screenSize; i++)
        {
            nib_put_scr(i, 0, '-');
            nib_put_scr(i, screenSize, '-');
            nib_put_scr(0, i, '|');
            nib_put_scr(screenSize, i, '|');
        }

        /*Get input while sleeping*/
        for (i = 0; i < 10; i++)
        {
            /*Get input*/
            input = nib_poll_kbd();
            if (input != -1)
            {
                if (input == 261)
                {
                    direction = 0;
                }
                else if (input == 259)
                {
                    direction = 1;
                }
                else if (input == 260)
                {
                    direction = 2;
                }
                else if (input == 258)
                {
                    direction = 3;
                }
                break;
            }
            /*Sleep*/
            usleep(sleepTime);
        }
        usleep(sleepTime*(10-i));
        clear();
}
nib_end();}
```