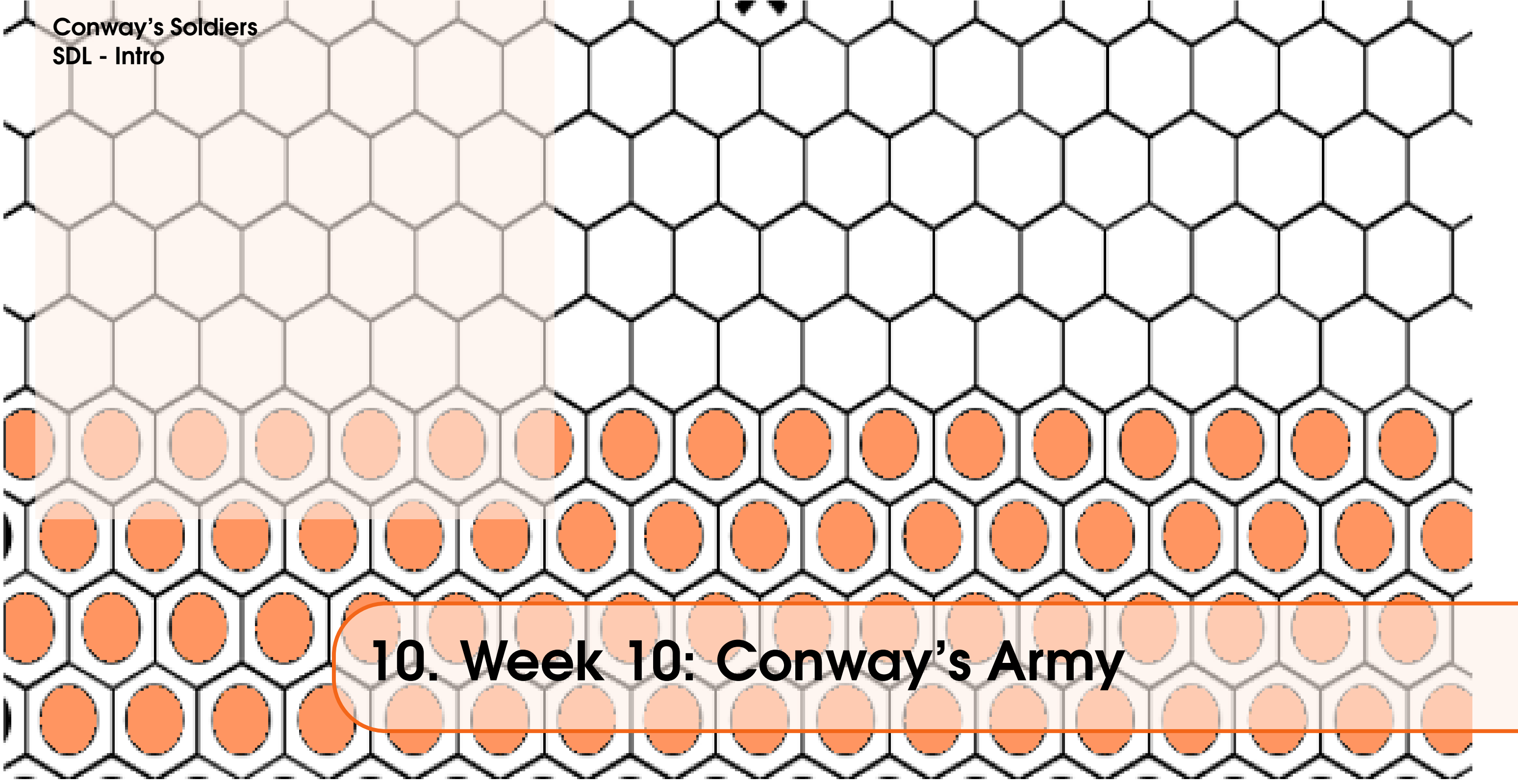

10. Week 10: Conway's Army

10.1 Conway's Soldiers

The one player game, *Conway's Soldiers* (sometimes known as *Solitaire Army*), is similar to peg solitaire. For this exercise, Conway's board is a 7 (width) $\times$ 8 (height) board with tiles on it. The lower half of the board is entirely filled with tiles (pegs), and the upper half is completely empty. A tile can move by jumping another tile, either horizontally or vertically (but never diagonally) onto an empty square. The jumped tile is then removed from the board.

The user enters the location of an empty square they'd like to get a tile into, and the program demonstrates the moves that enables the tile to reach there (or warns them it's impossible). To do this you will use a list of boards. The initial board is put into this list. Each board in the list is, in turn, read from the list and all possible moves from that board added into the list. The next board is taken, and all its resulting boards are added, and so on. However, one problem with is that repeated boards may be put into the queue and cycles occur. This soon creates an explosively large number of boards (several million). You can solve this by only adding a board into the list if an identical one has never been put into the list before. A linear search is acceptable for this task. Each structure in the list will contain (amongst other things) a board and a record of its parent board, i.e. the board that it was created from.

Exercise 10.1 Write a program that:
- Inputs a target location for a tile to reach (x in argv[1], y in argv[2]).
- Demonstrates the correct solution (reverse order is fine) using plain text or ncurses.

Use the algorithm described above and not anything else.

10.2 SDL - Intro

Many programming languages have no inherent graphics capabilities. To get windows to appear on the screen, or to draw lines and shapes, you need to make use of an external library. Here we use SDL[1], a cross-platform library providing the user with (amongst other things) such graphical capabilities.

[1] actually, we are using the most recent version SDL2, which is installed on all the lab machines

www `https://www.libsdl.org/`

The use of SDL is, unsurprisingly, non-trival, so some simple wrapper files have been created (`neillsdl2.c` and `neillsdl2.h`). These give you some simple functions to initialise a window, draw rectangles, wait for the user to press a key etc. and are somewhat similar to `neillncurses.*`.

An example program using this functionality is provided in a file `blocks.c`, shown in Figure 10.1.

This program initialises a window, then sits in a loop, drawing randomly positioned and coloured squares, until the user presses the mouse or a key.

Using the `Makefile` provided, compile and run this program.

SDL is already installed on lab machines. At home, if you’re using a ubuntu-style linux machine, use: `sudo apt-get install libsdl2-dev` to install it.

Exercise 10.2 Rewrite Conway’s Soldiers so that the solution is displayed using SDL. ■

```
#include <stdio.h>
#include <stdlib.h>
#include "neillsdl2.h"

#define RECTSIZE 20
#define MILLISECONDDELAY 20

int main(void)
{

   SDL_Simplewin sw;
   SDL_Rect rectangle;
   rectangle.w = RECTSIZE;
   rectangle.h = RECTSIZE;

   Neill_SDL_Init(&sw);
   do{

      SDL_Delay(MILLISECONDDELAY);

      /* Choose a random colour, a mixture of red, green and blue. */
      Neill_SDL_SetDrawColour(&sw,
                              rand()%SDL_8BITCOLOUR, rand()%SDL_8BITCOLOUR,
                              rand()%SDL_8BITCOLOUR);

      /* Filled Rectangle, fixed size, random position */
      rectangle.x = rand()%(WWIDTH-RECTSIZE);
      rectangle.y = rand()%(WHEIGHT-RECTSIZE);
      SDL_RenderFillRect(sw.renderer, &rectangle);

      /* Unfilled Rectangle, fixed size, random position */
      rectangle.x = rand()%(WWIDTH-RECTSIZE);
      rectangle.y = rand()%(WHEIGHT-RECTSIZE);
      SDL_RenderDrawRect(sw.renderer, &rectangle);

      Neill_SDL_UpdateScreen(&sw);

      /* Has anyone pressed ESC or killed the SDL window ?
         Must be called frequently - it's the only way of escaping */
      Neill_SDL_Events(&sw);

   }while(!sw.finished);


   /* Clear up graphics subsystems */
   atexit(SDL_Quit);

   return 0;

}
```

Figure 10.1: The program *blocks.c* used to demonstrate some SDL2 wrapper functions.