경북대학교
시스템프로그래밍 20162 A분반 2조

2015115907 김민석
2015112162 김철균
2015111075 정주영
2015112348 조현우

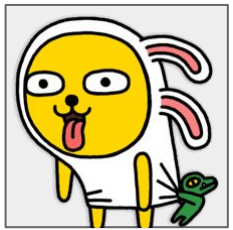
목차

1. 개요

- 조원 소개

김민석

김철균



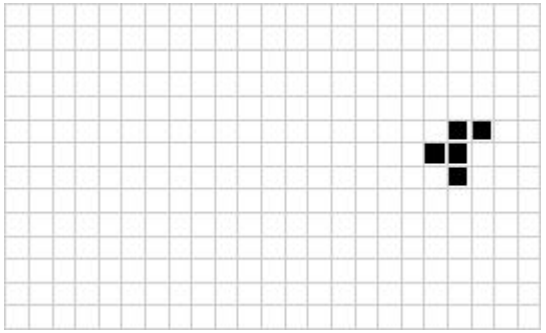
정주영

조현우

1. 개요

프로그램 제목 : Lifecraft

프로그램 간단 소개 : 콘웨이 생명게임 응용 ver

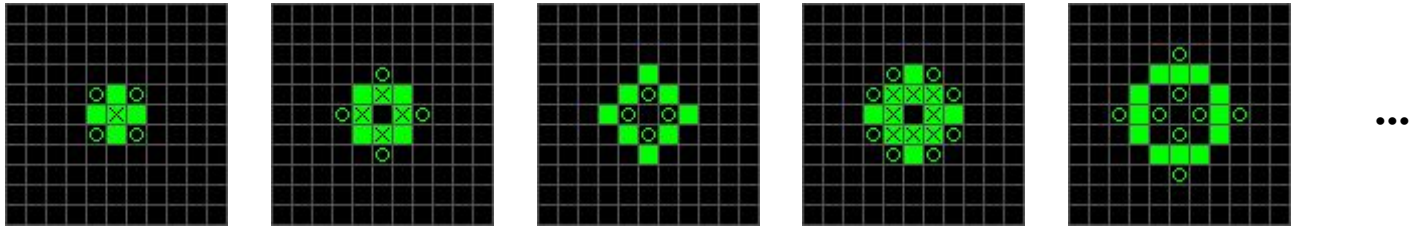


<기존 콘웨이 생명 게임>

- **2개의 진영** - Player 1, Player 2
- **3개의 유닛 종류** - Assassin (○,●)
  Bruiser (□,■),
  Commander (☆,★)
- **생명 탄생/소멸 조건은 기존의 콘웨이 생명 게임과 유사**

1. 개요

- 콘웨이 생명게임(Conway's Game of Life)이란?
  - 영국 수학자 존 호턴 콘웨이가 만든 오토마타(automata)의 일종
  - 평면 위에 세포가 놓여 있으며 각 세포의 초기 상태(삶, 죽음)에 따라 이후 상태가 계산된다.
  - 이후 상태를 계산하는 방법:
    - 세포가 살아 있을 때: 주변 8칸에 2~3개의 산 세포가 있다면 삶을 유지
    - 세포가 죽어 있을 때: 주변 8칸에 3개의 산 세포가 있다면 살아남



초기 상태 1세대 2세대 3세대 4세대

2. 사용 방법

- 실행 방법 : /게임파일 경로/lifecraft
- 1인 모드와 2인 모드가 존재
  - 1인 모드 : 컴퓨터와 플레이어
  - 2인 모드 : 플레이어 1과 플레이어 2
- 메인화면의 종료 메뉴를 선택하거나 Q를 입력하면 프로그램 종료

2. 사용 방법

- 커서 이동 : 방향키
- 유닛 배치 : 원하는 장소에 커서를 위치시킨 후 배치하고자 하는 유닛의 키를 입력
  - Assassin : a / Bruiser : b / Commander : c / 실행 취소 : u
- 유닛 배치 종료 : Return
- 일시 정지 : p

esc키를 입력하면 일시정지 메뉴가 표시됨

- Resume
- Restart
- Back to Main Menu

2. 사용 방법

- 게임 규칙

	Bruiser (□, ■)	Assassin (○,●)	Commander (☆,★)
점수	1	2	3
공격력	1	3	2
방어력	3	1	2

- **탄생할 조건**: 주변 8칸의 두 팀의 점수 합을 비교해 높은 팀의 유닛으로 생성된다. 이 때 점수 합의 차가 2점이면 Bruiser, 3점이면 Assassin, 4점이면 Commander가 탄생한다.
- **소멸할 조건**: 주변 8칸의 적 유닛의 공격력 합이 자신의 방어력 이상이면 소멸한다.

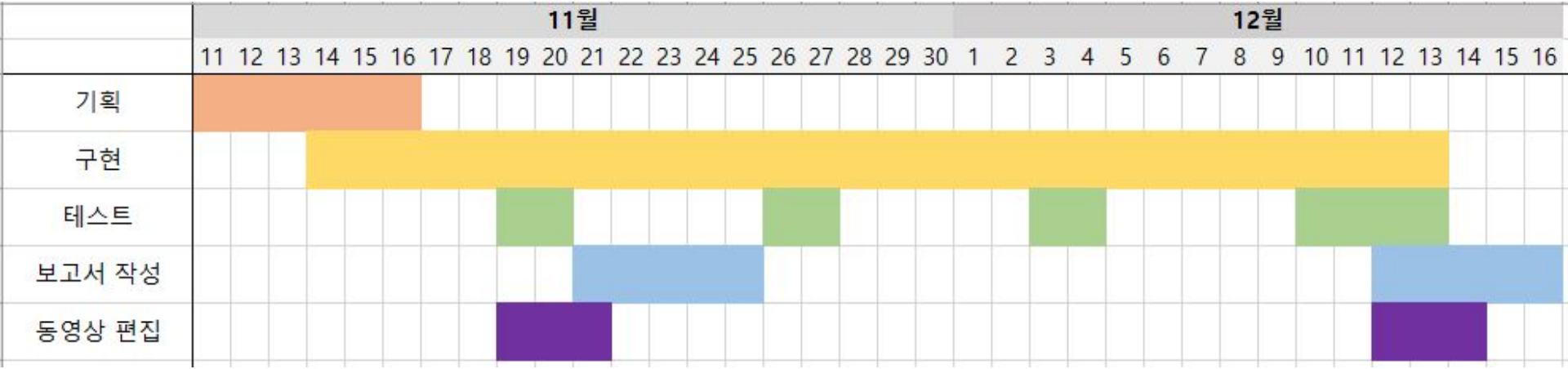
2. 사용 방법

- 게임 규칙

1. 플레이어는 각 유닛을 배치할 수 있는 최대 횟수와 종료 턴 수를 지정 후 게임을 시작한다.
2. 게임이 시작되면 각 플레이어가 번갈아가며 유닛을 배치한다.
3. 배치가 완료되면 플레이 버튼을 눌러 시뮬레이션을 시작한다.
4. 매 턴마다 유닛 탄생 조건과 소멸 조건을 계산한다.
5. 한 쪽의 유닛이 모두 소멸하거나 지정한 종료 턴 수에 도달한 경우 점수가 높은 플레이어가 승리한다.

3. 계획

- 계획표


11월
12월
11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
기획
구현
테스트
보고서 작성
동영상 편집

4. 구현

- 데모 동영상 (중간)
  - https://youtu.be/AZ5BCslzshw

4. 구현

- 주요 코드 설명 - void **dress**(d_code code, char *msg, ...)
  - **ANSI escape 코드**를 앞에 붙이고 메시지 출력
  - 글자에 서식을 넣거나 커서 상태를 바꾸려 할 때 사용
  - msg에서 %d, %s 등의 포맷 문자열 사용 가능
  - 인자
    - d_code code : 적용할 서식의 종류
    - char *msg : 메시지
    - … : 메시지에 표시될 정보
  - ANSI escape 코드란?
    - 커서나 터미널의 출력 방식을 변경할 수 있게 해주는 코드. 주로 ASCII 27번 문자(ESC)를 이용한다.

```
case BOLD: strcat(buf, "1m"); break;
case UNDERLINE: strcat(buf, "4m"); break;

case F_BLACK: strcat(buf, "30m"); break;
case F_RED: strcat(buf, "31m"); break;
case F_GREEN: strcat(buf, "32m"); break;
case F_YELLOW: strcat(buf, "33m"); break;
case F_BLUE: strcat(buf, "34m"); break;
case F_MAGENTA: strcat(buf, "35m"); break;
case F_CYAN: strcat(buf, "36m"); break;
case F_WHITE: strcat(buf, "37m"); break;

case B_BLACK: strcat(buf, "40m"); break;
case B_RED: strcat(buf, "41m"); break;
case B_GREEN: strcat(buf, "42m"); break;
case B_YELLOW: strcat(buf, "43m"); break;
```

4. 구현

- 주요 코드 설명 - void **evolve**(void *_b, int rows, int cols)
  - 게임 진행 중 유닛의 생사를 결정하는 함수
  - 인자
    - void *_b : 현재 보드
    - int rows : 보드의 행 수
    - int cols : 보드의 열 수

```
// 주변 유닛 수 계산 (음수는 플레이어1, 양수는 플레이어2)
for(y=r-1; y<=r+1; y++){
    for(x=c-1; x<=c+1; x++){
        sur = (int)board[(y+rows)%rows][(x+cols)%cols];
        n += cell_sc[sur];
        if(t + cell_ow[sur] == 0) as += cell_at[sur];
    }
}
```

  - 인자로 받은 현재 보드(void *_b)를 한칸씩 확인하며 게임 규칙에 따라 생사를 결정
  - cell_type get_evolved_cell(int n) : 유닛 주변의 점수합을 인자로 받아 새로 생겨날 유닛의 종류를 반환
  - int cell_sc[], int cell_hp[], int cell_at[], int cell_ow[] : 유닛의 점수, 방어력(체력), 공격력, 소유자를 저장하는 배열
  - int n : 현재 유닛 주변 아군 유닛의 점수합

4. 구현

- 주요 코드 설명 - void **evolve**(void *_b, int rows, int cols)
  - int as, int t : 상대 유닛의 공격력 합, 해당 유닛의 소유자
  - 게임 규칙에 따라 생사를 결정한 후 임시 보드에 저장
  - 이번 턴의 생사 결정이 완료되면 원래 보드인 void *_b에 임시 보드(cell_type new[rows][cols])의 내용을 복사

```
// 생사 결정
if(board[r][c] == CT_NONE){
    new[r][c] = get_evolved_cell(n);
}else if(hp <= as){
    new[r][c] = CT_NONE;
}else{
    new[r][c] = board[r][c];
}
```

```
//원래 보드에 이번 턴의 생사 결정이 완료된 임시 보드를 복사
for(r=0; r<rows; r++){
    for(c=0; c<cols; c++){
        board[r][c] = new[r][c];
    }
}
```

4. 구현

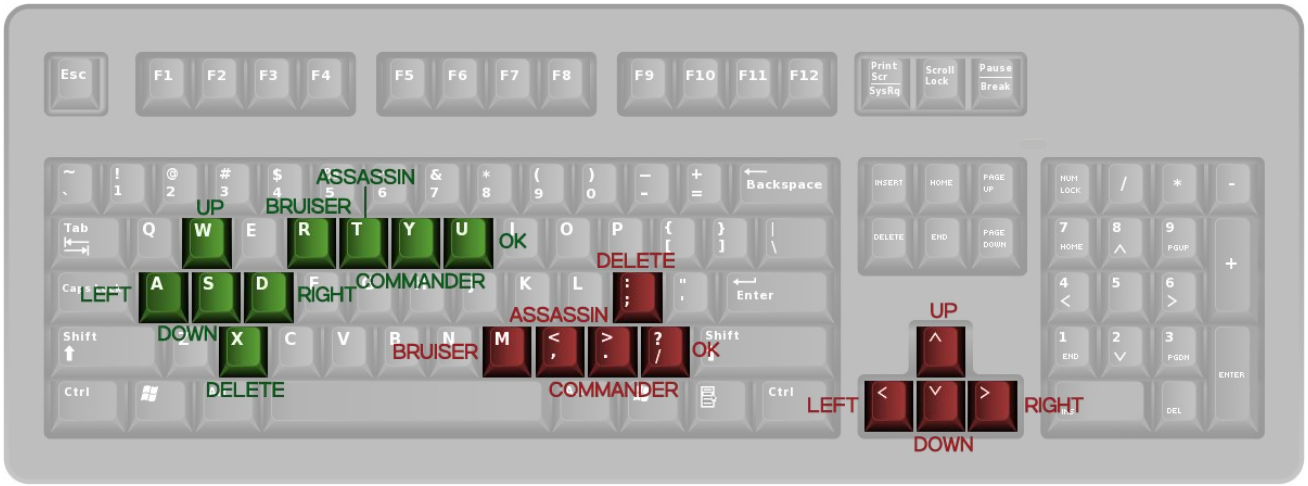
- 주요 코드 설명 - void* **key_manage**(void *arg)
  - 키 입력을 받아 처리하는 함수
  - 입력 키 처리 방식
    - main에서 key_manage 쓰레드 생성후 키입력시 main 쓰레드 대기
    - key_manage함수에서 키 입력을 받고 전역변수 key에 저장후 main으로 signal을 보냄
    - main 쓰레드 대기상태 해제후 prom_~ 함수에서 입력받은 전역변수 key를 사용

```
while(1){
        key = getchar() & 0xFF;
        if(key == 27){
                key |= (getchar() & 0xFF) << 8;
                if(key == 0x5B1B){
                        key |= (getchar() & 0xFF) << 16;
                }
        }
        pthread_cond_signal(&main_cond);
        // printf("Key: %d %d %d\n", key >> 16, key >> 8 & 0xFF, key & 0xFF);
}
```

- 지금은 Unit을 다 배치해야만 넘어가도록 구현 -> 타임오버 또는 두 플레이어가 완료키 누를 때 넘어가도록 수정

4. 구현

- 주요 코드 설명 - void* key_manage(void *arg)
  - 작동 키 : player 1 : 초록 player 2 : 빨강



4. 구현

```
void ui(){
        cell_type board[R][C] = { CT_NONE, };
        void (*menu_items[])(void*) = { NULL, menu_title, menu_ready, menu_go, menu_result, menu_help, NULL };
        menu (*prom_items[])(void*) = { NULL, prom_title, prom_ready, prom_go, prom_result, prom_help, NULL };
        menu state = TITLE, next;

        while(1){
                DRESS_NEW;
                if(state == END) break;
                menu_items[state](board);
                printf("\n");
                pthread_cond_wait(&main_cond, &main_lock);
                if((next = prom_items[state](board)) != MENU_NONE) state = next;
        }
}
```

- 주요 코드 설명 - void **ui**()
  - menu와 prom의 수준별 관리를 위해 포인터 배열을 선언
  - state별 menu_~함수를 호출해서 콘솔 출력후 key_manage에 의한 콘솔 입력을 기다림
  - state별 prom_~함수를 호출해서 전역변수로 선언된 key를 사용해 알맞은 기능을 수행
  - prom_~함수의 리턴값을 활용해 state수준을 변경시킴

4. 구현

- 데모 동영상 (최종)
  - https://youtu.be/DQ6oBBPm0Ww