

# Tetris Class 만들기

송실대학교  
김강희 교수  
([khkim@ssu.ac.kr](mailto:khkim@ssu.ac.kr))

# 순서

## ❖ 이론:

- main() 위주로 작성된 코드를 Tetris 클래스로 구성
- 상태 기계 (Finite State Machine)

## ❖ 실습:

- Tetris 클래스 작성
- Tetris 클래스 검증 (복수 객체)

# Tetris Class 를 만들자

- ❖ 이제 Tetris class 를 만들 준비가 되었다. 이유는?
  - ❖ Tetris 객체를 상태 기계로 이해하자
    - 입력은 key 값, 출력은 oScreen 이다.
    - object.accept(key) → object.oScreen
  - ❖ Tetris 객체(데이터 모델)는 deterministic 해야 한다.
    - 동일한 입력 시퀀스에 대해서 동일한 출력 시퀀스를 얻어야 한다.
    - 그러자면, 난수 발생을 객체 외부에서 제공해야 한다.
    - object.accept(rand\_num) → object.oScreen
  - ❖ 하나의 상태 기계는 개념적으로 하나의 입력 함수를 가져야 한다.
    - accept(key)와 accept(rand\_num)을 하나의 함수로 표현한다.
  - ❖ static/dynamic 변수들을 정의하고 초기화 함수를 정의한다.
  - ❖ private/public 변수들을 정의하고 초기화 함수를 정의한다.
  - ❖ hardcoded constant 들을 제거하자 → enum type 사용
- 위 조건들을 모두 만족하는 Tetris class 를 작성했다고 가정하고, 이를 사용하는 main 함수를 먼저 작성하자.

# 상태 기계

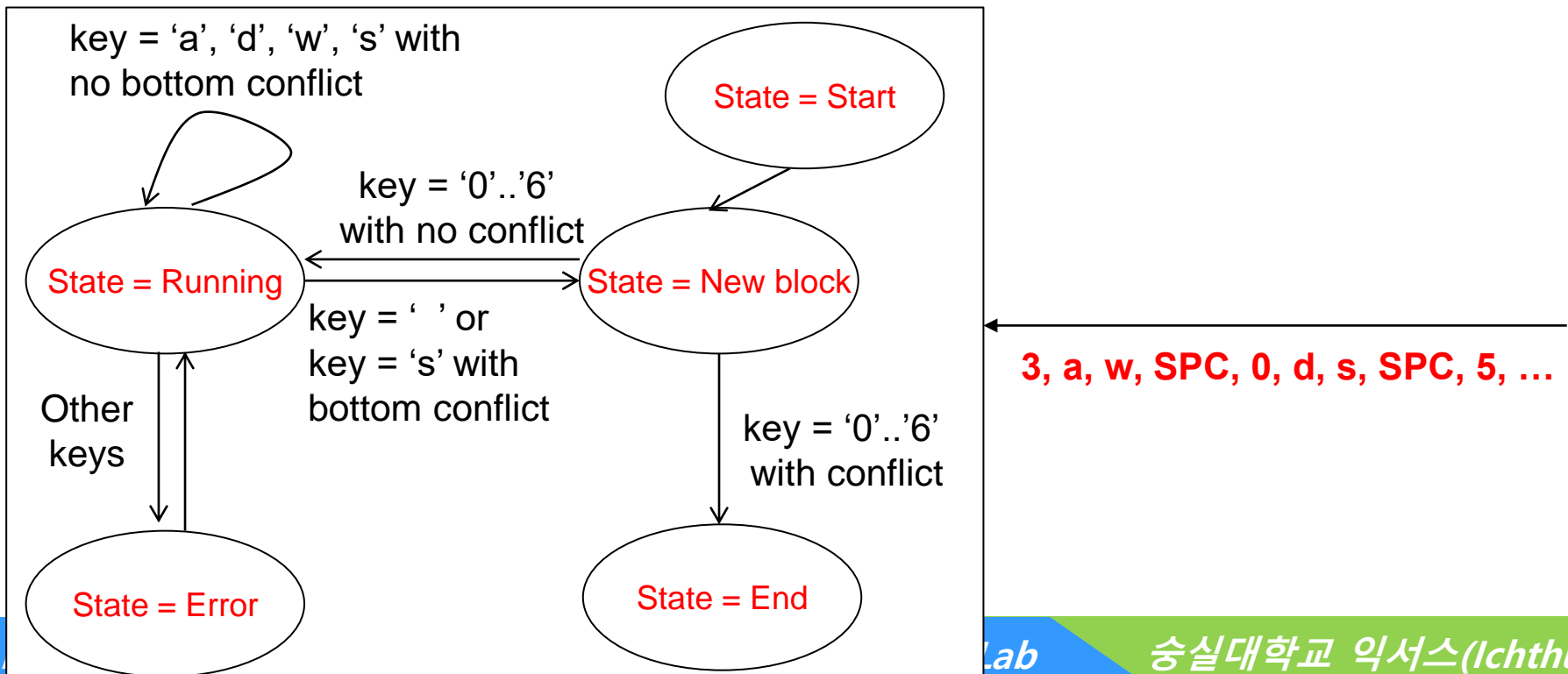
## ❖ 상태 기계

- 유한 상태 기계(finite-state machine, FSM) 또는 유한 오토마톤(finite automaton, 복수형: 유한 오토마타 finite automata)이라고 번역함
- 컴퓨터 프로그램과 전자 논리 회로를 설계하는데 쓰이는 수학적 모델로서 간단히 상태 기계라고 부르기도 함
- 유한 상태 기계는 유한한 개수의 상태를 가질 수 있는 오토마타, 즉 추상 기계라고 할 수 있음
- 한 번에 오로지 하나의 상태만을 가지게 됨
- 현재 상태(current state)란 현재 시간의 상태를 지칭함
- 어떠한 사건(event)에 의해 한 상태에서 다른 상태로 변화할 수 있으며, 이를 전이(transition)이라 함
- 유한 상태 기계는 현재 상태에서부터 가능한 전이 상태와 이러한 전이를 유발하는 조건들의 집합으로서 정의됨

# 상태 기계

## ❖ 테트리스 게임의 상태 기계 모델

- key 값과 idxType 값의 나열을 하나의 입력 시퀀스(input sequence)로 이해하고 Tetris 상태 기계는 Start, Running, New Block, End, Error 상태를 가진다고 이해할 수 있음
- 동일한 입력 시퀀스에 대해서 상태 기계는 항상 동일한 내부 상태를 가짐
- Tetris 상태 기계의 입력을 하나의 변수 타입으로 통일하면 Tetris class 코드 상에 상태 기계 관점을 더 잘 표현할 수 있음



# Tetris 클래스를 사용하는 Main 함수

```
180 int main(int argc, char *argv[]) {
181     char key;
182     registerAlarm(); // register one-second timer
183     srand((unsigned int)time(NULL)); // init the random number generator
184
185     TetrisState state;
186     Tetris::init(setOfBlockArrays, MAX_BLK_TYPES, MAX_BLK_DEGREES); // static 변수들 초기화
187     Tetris *board = new Tetris(10, 10); // dynamic 변수들 초기화
188     key = (char) ('0' + rand() % board->get_numTypes()); // rand_num 를 char 형으로 변환함
189     board->accept(key);
190     drawScreen(board->get_oScreen(), board->get_wallDepth()); cout << endl;
191
192     while ((key = getch()) != 'q') {
193         state = board->accept(key);
194         drawScreen(board->get_oScreen(), board->get_wallDepth()); cout << endl;
195         if (state == TetrisState::NewBlock) {
196             key = (char) ('0' + rand() % board->get_numTypes());
197             state = board->accept(key);
198             drawScreen(board->get_oScreen(), board->get_wallDepth()); cout << endl;
199             if (state == TetrisState::Finished)
200                 break;
201         }
202     }
203
204     delete board;
205     Tetris::deinit();
206     cout << "(nAlloc, nFree) = (" << Matrix::get_nAlloc() << ',' << Matrix::get_nFree() << ")" << endl;
207     cout << "Program terminated!" << endl;
208     return 0;
209 }
```

// 다음 accept()가 이동 key 를 원하는지, rand\_num 을 원하는지 구분하기 위해 accept() 리턴값을 정의함

# Tetris.h

// 자식 클래스도 사용하도록 **extern** 선언함

```
14 extern Matrix *deleteFullLines(Matrix *screen, Matrix *blk, int top, int dw);
15 //extern int *allocArrayScreen(int dy, int dx, int dw);
16 //extern void deallocArrayScreen(int *array);
```

class Tetris {  
protected: // 자식 클래스에게 상속되도록 **protected** 선언함

```
20 // static members
21 static Matrix ***setOfBlockObjects;
22 static int numTypes;
23 static int numDegrees;
24 static int wallDepth;
```

// dynamic members

```
27 int rows; // rows of screen = dy + wallDepth
28 int cols; // columns of screen = dx + 2*wallDepth
29 int type;
30 int degree;
31 int top;
32 int left;
```

```
34 TetrisState state;
35 Matrix *iScreen;
36 Matrix *oScreen;
37 Matrix *currBlk;
```

// 상태 변수들

```
1 #pragma once
2 #include <iostream>
3 #include <cstdlib>
4 #include "Matrix.h"
5
6 using namespace std;
7
8 enum class TetrisState {
9     NewBlock,
10    Running,
11    Finished,
12 };
13
```

# Tetris.h

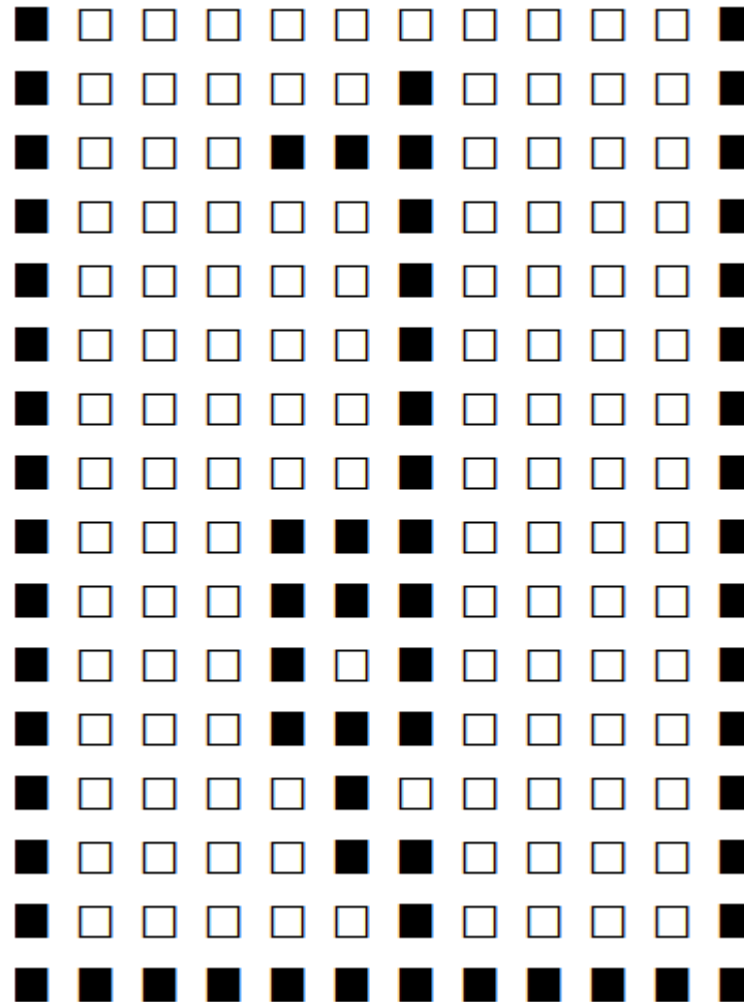
```
39 public:
40     static void init(int **setOfBlockArrays, int nTypes, int nDegrees);
41     static void deinit(void);
42     Tetris(int cy, int cx);
43     ~Tetris();
44
45     // accessors
46     static int get_wallDepth(void) { return wallDepth; }
47     static int get_numTypes(void) { return numTypes; }
48     Matrix *get_oScreen(void) const { return oScreen; }
49
50     // mutators
51     TetrisState accept(char key);
52 };
```



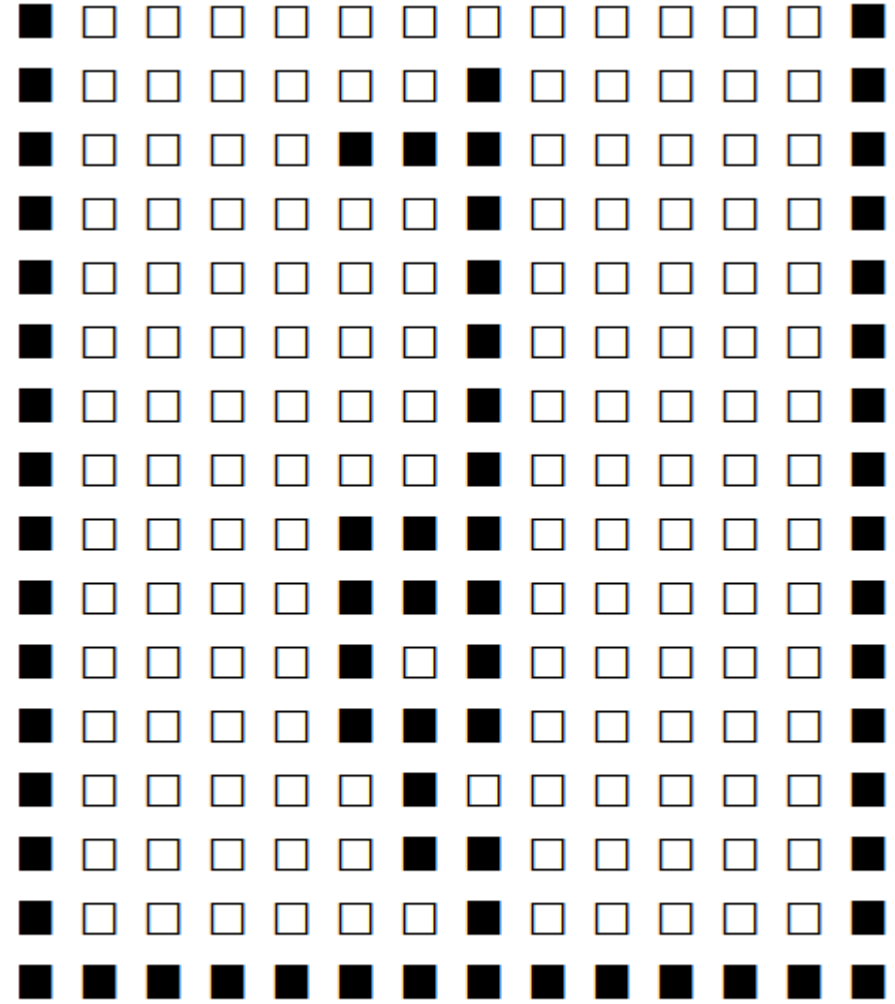
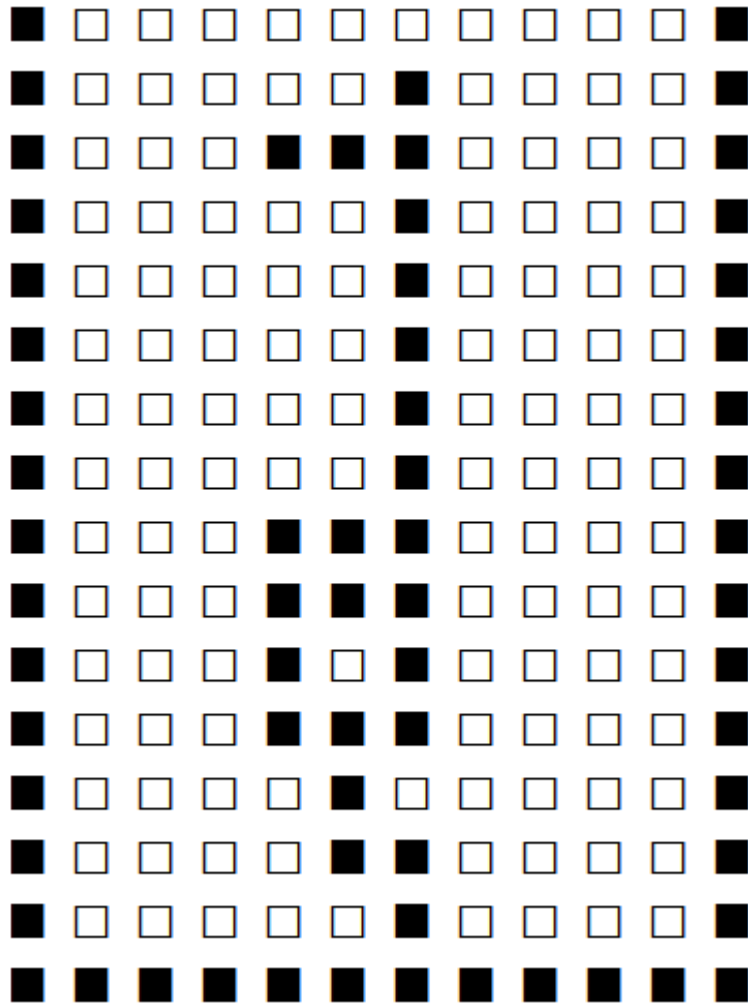
# Tetris.cpp

```
146 TetrisState Tetris::accept(char key) {
147
148     if (state == TetrisState::Finished)
149         return state;
150
151     else if (state == TetrisState::NewBlock) {
152
153         int idx = key - '0';
154         state = TetrisState::Running;
155
156         ...
157
158         if (tempBlk2->anyGreaterThan(1)) // exit the game
159             state = TetrisState::Finished;
160         delete tempBlk2;
161
162         return state; // = Running or Finished
163     }
164     else if (state == TetrisState::Running) {
165
166         state = TetrisState::Running;
167         bool touchDown = false;
168
169         ...
170
171         if (touchDown) {
172             oScreen = deleteFullLines(oScreen, currBlk, top, wallDepth);
173             iScreen->paste(oScreen, 0, 0);
174             state = TetrisState::NewBlock;
175         }
176
177         return state; // = Running or NewBlock
178     }
179
180     return state; // not reachable
181 }
```

# 단일 객체



# 다중 객체: static/dynamic 변수 분리 확인



Program terminated!

|1

# 남은 숙제

- ❖ 다음 조건을 만족하는 CTetris class를 Tetris class를 상속받아 작성할 것
  - Tetris.h & Tetris.cpp 는 수정해서는 안 됨
  - 다음 페이지의 main 함수가 그대로 실행되어야 함.
  - 빈칸은 0, 벽은 1로 표현할 것
  - main 함수의 setOfColorBlockArrays 배열에서
    - ❖ 블록 1은 0 아닌 부분을 10으로 표현함
    - ❖ 블록 2는 0 아닌 부분을 20으로 표현함
    - ❖ ...
    - ❖ 블록 7은 0 아닌 부분을 70으로 표현함
  - 각 블록은 drawScreen() 함수에 의해서 오른쪽 화면과 같이 서로 다른 글자 모양으로 출력됨

```
X □ □ □ ♣ □ □ □ □ □ □ X
X □ □ □ ♣ ♣ ♥ □ □ □ □ □ X
X □ □ □ □ □ ♠ □ □ □ □ □ X
X □ □ □ □ □ ♠ □ □ □ □ □ X
X □ □ □ ♣ □ ♠ □ □ □ □ □ X
X □ □ □ ♣ ♣ ♣ □ □ □ □ □ X
X □ □ □ □ ▲ □ □ □ □ □ □ X
X □ □ □ ▲ ▲ □ □ □ □ □ □ X
X □ □ □ ▲ □ □ □ □ □ □ □ X
X □ □ □ ● ● □ □ □ □ □ □ X
X □ □ □ ● ● □ □ □ □ □ □ X
X □ □ □ □ ▼ □ □ □ □ □ □ X
X □ □ □ □ ▼ ▼ □ □ □ □ □ □ X
X □ □ □ □ ■ ▼ □ □ □ □ □ □ X
X □ □ □ ■ ■ ■ □ □ □ □ □ □ X
X X X X X X X X X X X X X
```

# 남은 숙제: CTetris 를 위한 main 함수

```
180 int main(int argc, char *argv[]) {
181     char key;
182     registerAlarm(); // register one-second timer
183     srand((unsigned int)time(NULL)); // init the random number generator
184
185     TetrisState state;
186     CTetris::init(setOfColorBlockArrays, MAX_BLK_TYPES, MAX_BLK_DEGREES);
187     CTetris *board = new CTetris(10, 10);
188     key = (char) ('0' + rand() % board->get_numTypes());
189     board->accept(key);
190     drawScreen(board->get_oCScreen(), board->get_wallDepth()); cout << endl;
191
192     while ((key = getch()) != 'q') {
193         state = board->accept(key);
194         drawScreen(board->get_oCScreen(), board->get_wallDepth()); cout << endl;
195         if (state == TetrisState::NewBlock) {
196             key = (char) ('0' + rand() % board->get_numTypes());
197             state = board->accept(key);
198             drawScreen(board->get_oCScreen(), board->get_wallDepth()); cout << endl;
199             if (state == TetrisState::Finished)
200                 break;
201         }
202     }
203
204     delete board;
205     CTetris::deinit();
206     cout << "(nAlloc, nFree) = (" << Matrix::get_nAlloc() << ',' << Matrix::get_nFree() << ")" << endl;
207     cout << "Program terminated!" << endl;
208     return 0;
209 }
```

# 남은 숙제: CTetris.h

❖ CTetris.h 파일은 변경하지 말 것!

```
1  #pragma once
2  #include <iostream>
3  #include <cstdlib>
4  #include "Tetris.h"
5
6  using namespace std;
7
8  class CTetris : public Tetris {
9  private:
10     // static members
11     static Matrix ***setOfColorBlockObjects;
12
13     // dynamic members
14     Matrix *iCScreen;
15     Matrix *oCScreen;
16     Matrix *currCBlk;
17
18 public:
19     static void init(int **setOfColorBlockArrays, int nTypes, int nDegrees);
20     static void deinit(void);
21     CTetris(int cy, int cx);
22     ~CTetris();
23
24     // accessors
25     Matrix *get_oCScreen(void) const { return oCScreen; }
26
27     // mutators
28     TetrisState accept(char key); // 부모 클래스의 accept 함수를 override 함
29 };
```

# 남은 숙제: CTetris.cpp

- ❖ CTetris.cpp 파일은 컴파일 가능하도록 필요한 코드들이 모두 작성되어 있음
- ❖ 다음 두 개의 함수들은 미완성이니 완성하기 바람 → 가능한 한 적은 라인 개수로 작성할 것!!
  - CTetris::init()
    - ❖ 이 함수 안에서 **Tetris::init()** 호출하는 코드는 수정하지 말 것
    - ❖ Tetris::init() 함수는 setOfBlockArrays 배열이 0과 1 이외의 원소를 갖는 경우에 0 아닌 원소들을 자동으로 1로 변환하는 기능을 가지고 있음
  - CTetris::accept()
    - ❖ 이 함수 안에서 **Tetris::accept()** 호출하는 코드는 수정하지 말 것
    - ❖ 이 함수 안에서 필요시 다음 함수 호출을 사용할 수 있음
      - oCScreen = deleteFullLines(oCScreen, currCBlk, top, wallDepth);
      - **deleteFullLines()** 함수는 Tetris.cpp 에서 제공하는 non-member 함수로서 extern 선언되어 외부 파일에서 호출할 수 있음
      - deleteFullLines() 함수는 첫번째 인자가 0과 1 이외의 원소를 갖는 행렬일지라도 정상적으로 삭제 기능을 수행할 수 있음

# 남은 숙제: Makefile

```
1 # Set compiler to use
2 CC=g++
3 CFLAGS=-g -I. -fpermissive
4 LDFLAGS_TET=
5 DEPS_TET=CTetris.h Tetris.h Matrix.h
6 OBJS_TET=Main.o CTetris.o Tetris.o Matrix.o ttymodes.o
7 DEBUG=0
8
9 all:: Main.exe
10
11 Main.exe: $(OBJS_TET)
12     $(CC) -o $@ $^ $(CFLAGS) $(LDFLAGS_TET)
13
14 %.o: %.c $(DEPS_TET)
15     $(CC) -c -o $@ $< $(CFLAGS)
16
17 %.o: %.cpp $(DEPS_TET)
18     $(CC) -c -o $@ $< $(CFLAGS)
19
20 clean:
21     rm -f *.exe *.o *~ *.stackdump
```