C Language Study Game

C -JUMP

PL Kwon Yong Hyun
Jun Sang Gyun
Kim Sung Han
Kwon Duck Hyun

Contents

1 Motive

2 Scenario

3 Architecture

4 Schedule

5 Q n A

Motive

Motive

- What is a famous IT brand in Korea?
- So, What is a famous IT brand in U.S.A?

Why?

[Brand Image & Selling]

- This is a corporate image research.
- It is because a focus on enterprise survey.

Motive

- They are taught from childhood to IT education.

Interested in IT.

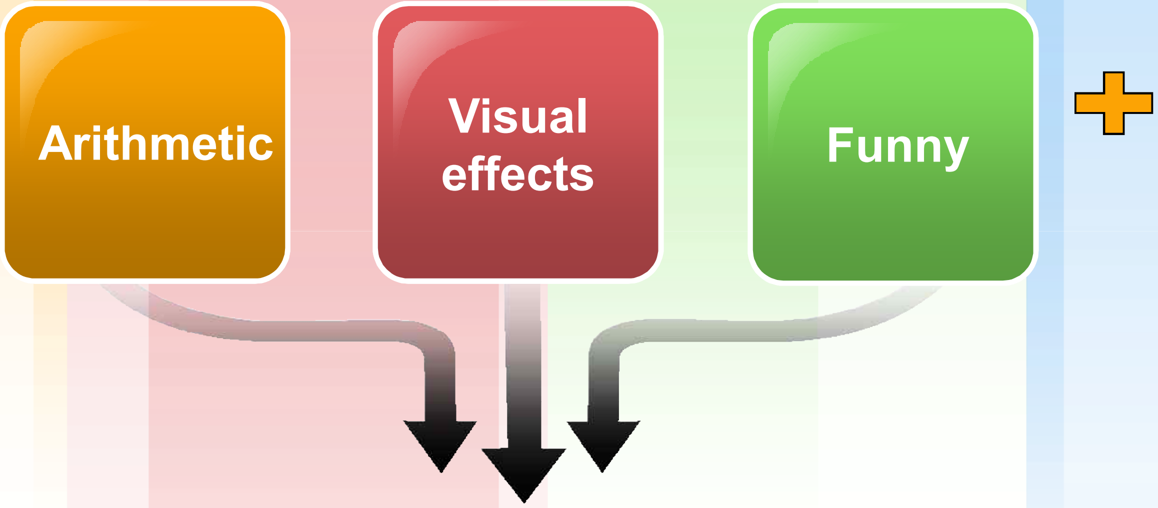
ut.

Content

- limited to English, mathematics, other Arts and Physical Education.

Motive

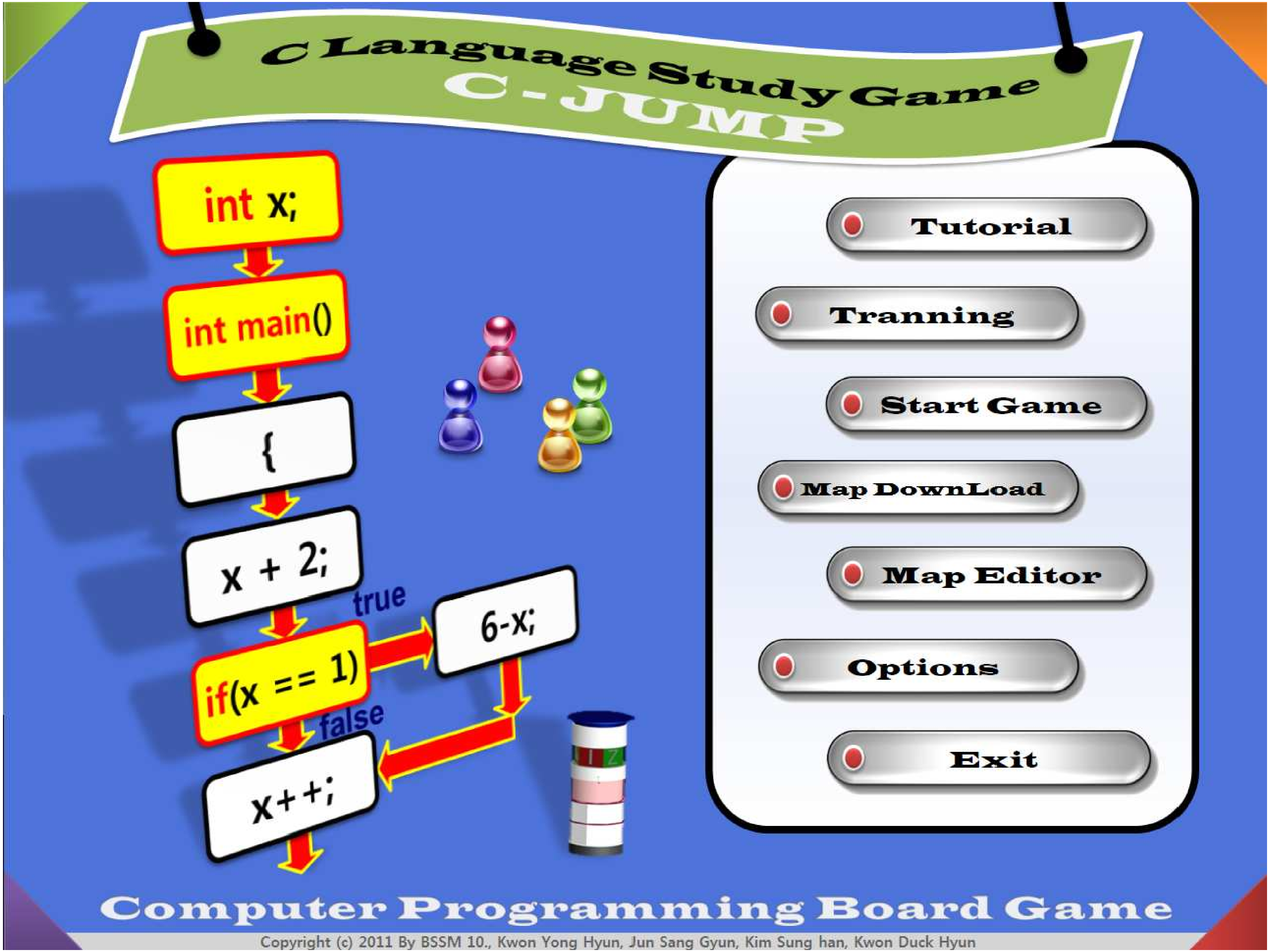
We need. Easy to learn, indirectly and Fun ways to learn



C Language Study Game

C - JUMP

Scenario



C Language Study Game

C-JUMP

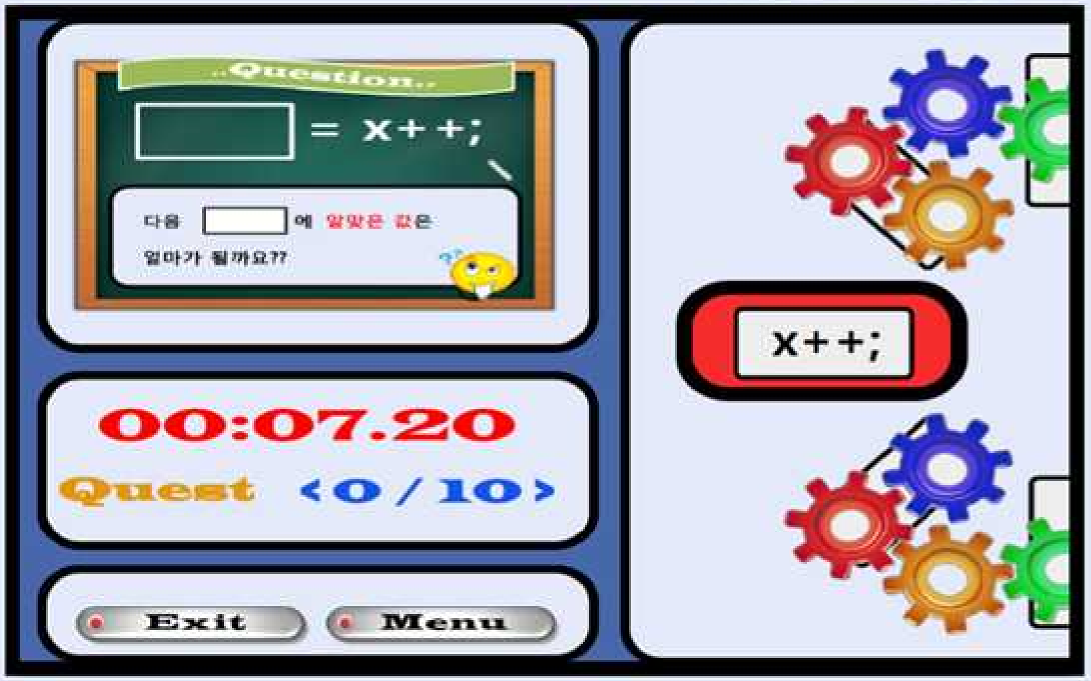
Computer Programming Board Game

MENU

- if
- else
- main
- x+3
- x++;
- x--;
- { }
- while
- switch
- continue

Back

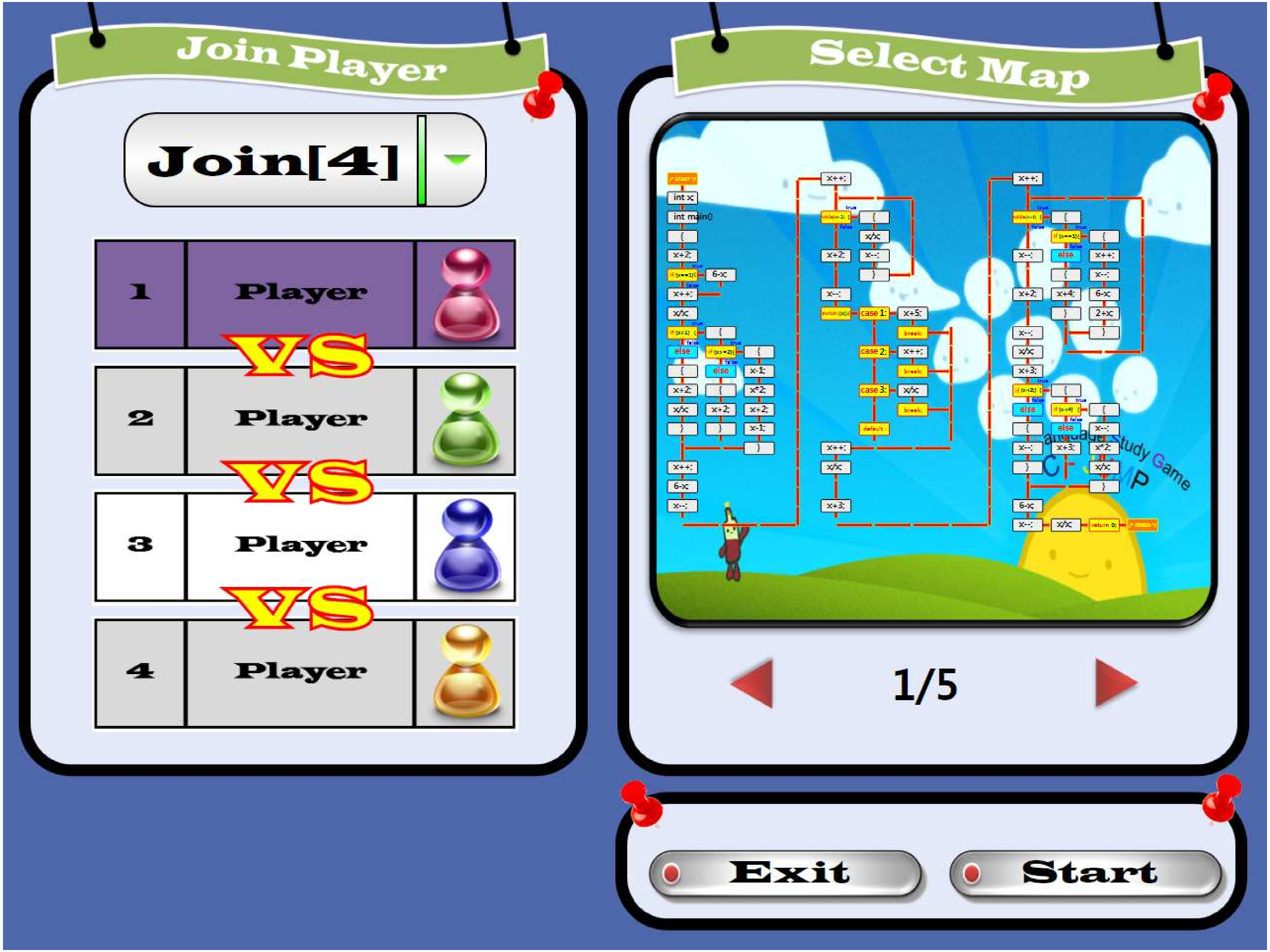
Introduction C-JUMP



다음 □ 에 알맞은 값은 얼마가 될까요??

1 / 5

C-Jump는 수학의 사칙연산과 시각적인 효과를 이용하여 C언어의 기초적인 이해와 관심을 높이는 BoardGame입니다.



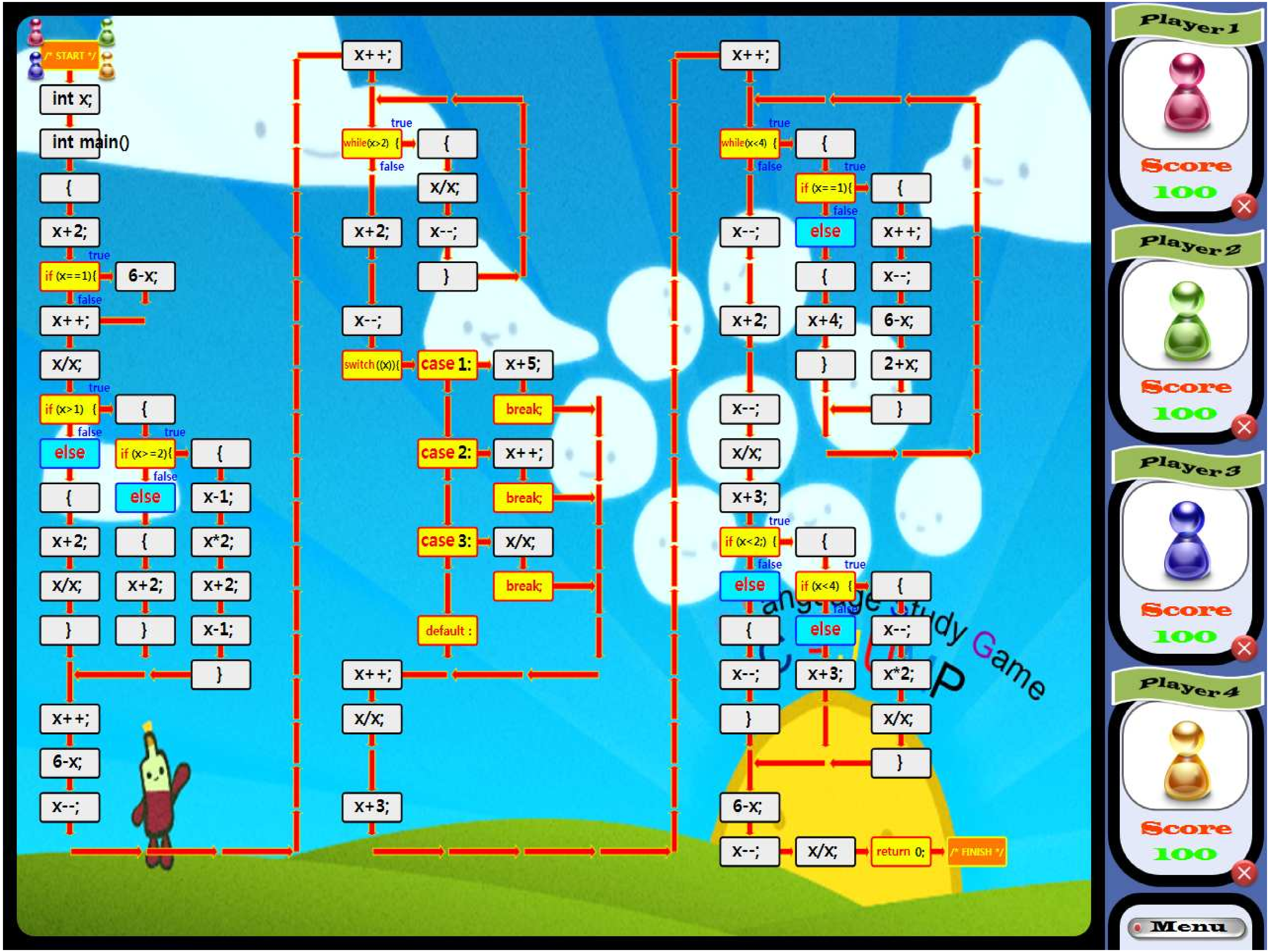
Join Player

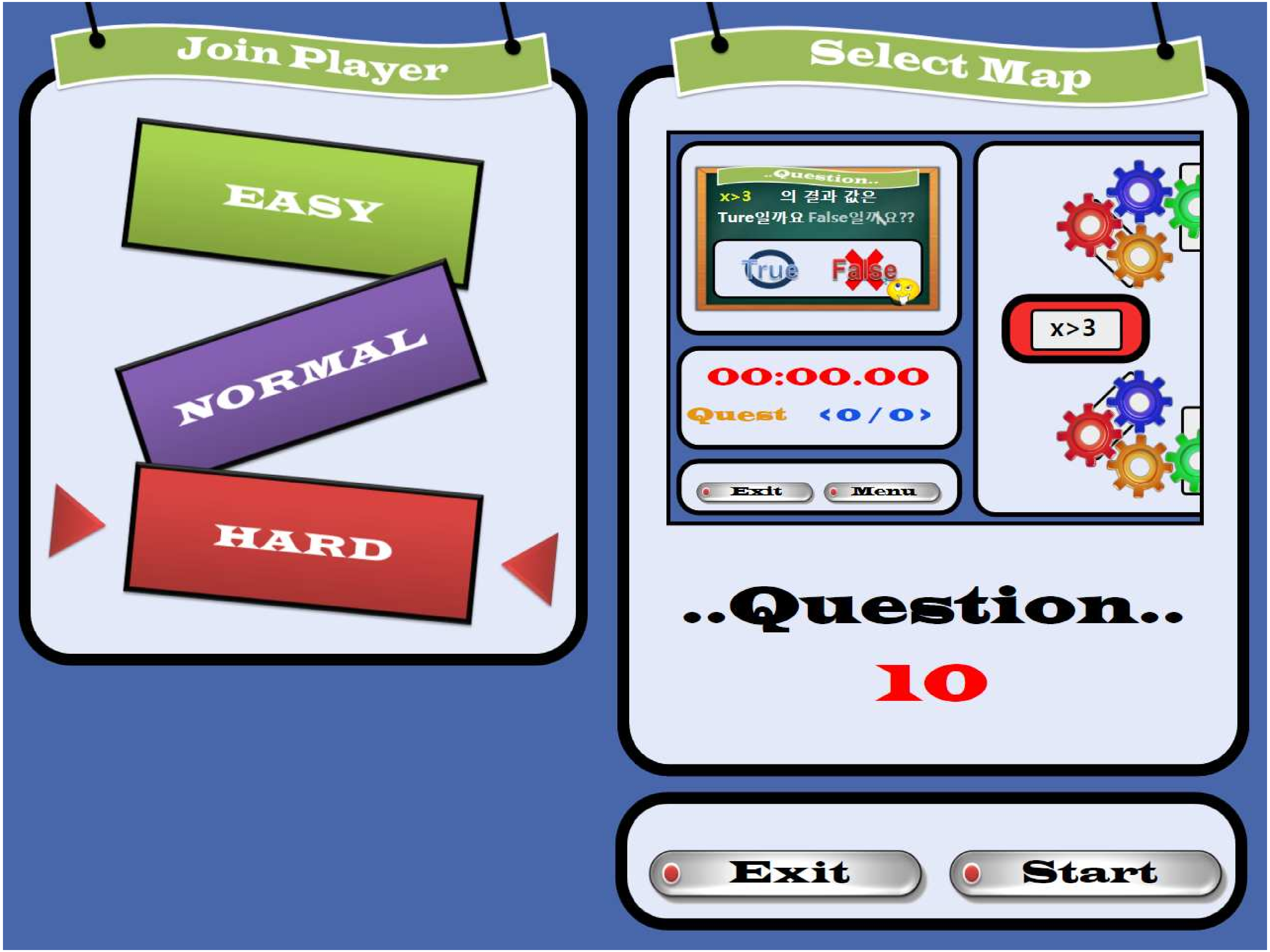
Join[4]

1	Player	
2	Player	
3	Player	
4	Player	

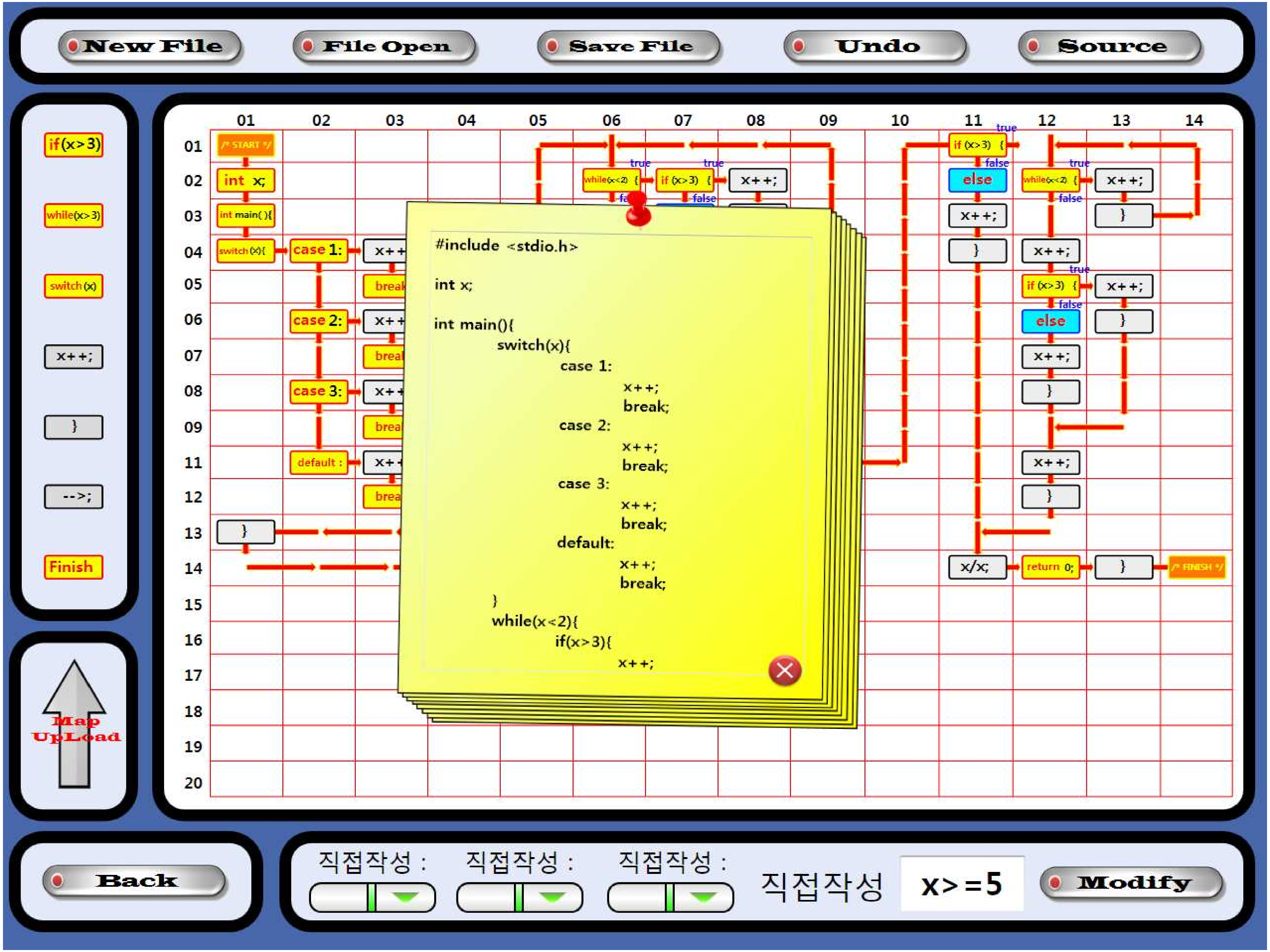
Select Map

1/5


/* START */
int x;
int main()
{
x+2;
if (x==1){
true
6-x;
false
x++;
x/x;
if (x>1) {
true
{
false
else
if (x>=2){
true
{
false
else
x-1;
{
{
x*2;
x+2;
x/x;
x+2;
x+2;
}
}
x-1;
}
x++;
6-x;
x--;
x++;
while(x>2) {
true
{
false
x/x;
x+2;
x--;
}
x--;
switch((x)){
case 1:
x+5;
break;
case 2:
x++;
break;
case 3:
x/x;
break;
default :
x++;
x/x;
x+3;
x++;
while(x<4) {
true
{
false
if (x==1){
true
{
false
x--;
else
x++;
{
x--;
x+2;
x+4;
6-x;
}
2+x;
x--;
}
x/x;
x+3;
if (x<2;) {
true
{
false
else
if (x<4) {
true
{
false
{
else
x--;
x--;
x+3;
x*2;
}
x/x;
}
6-x;
x--;
x/x;
return 0;
/* FINISH */
Language Study Game
Player 1
Score
100
Player 2
Score
100
Player 3
Score
100
Player 4
Score
100
Menu



x>3 의 결과 값은
Ture일까요 False일까요??

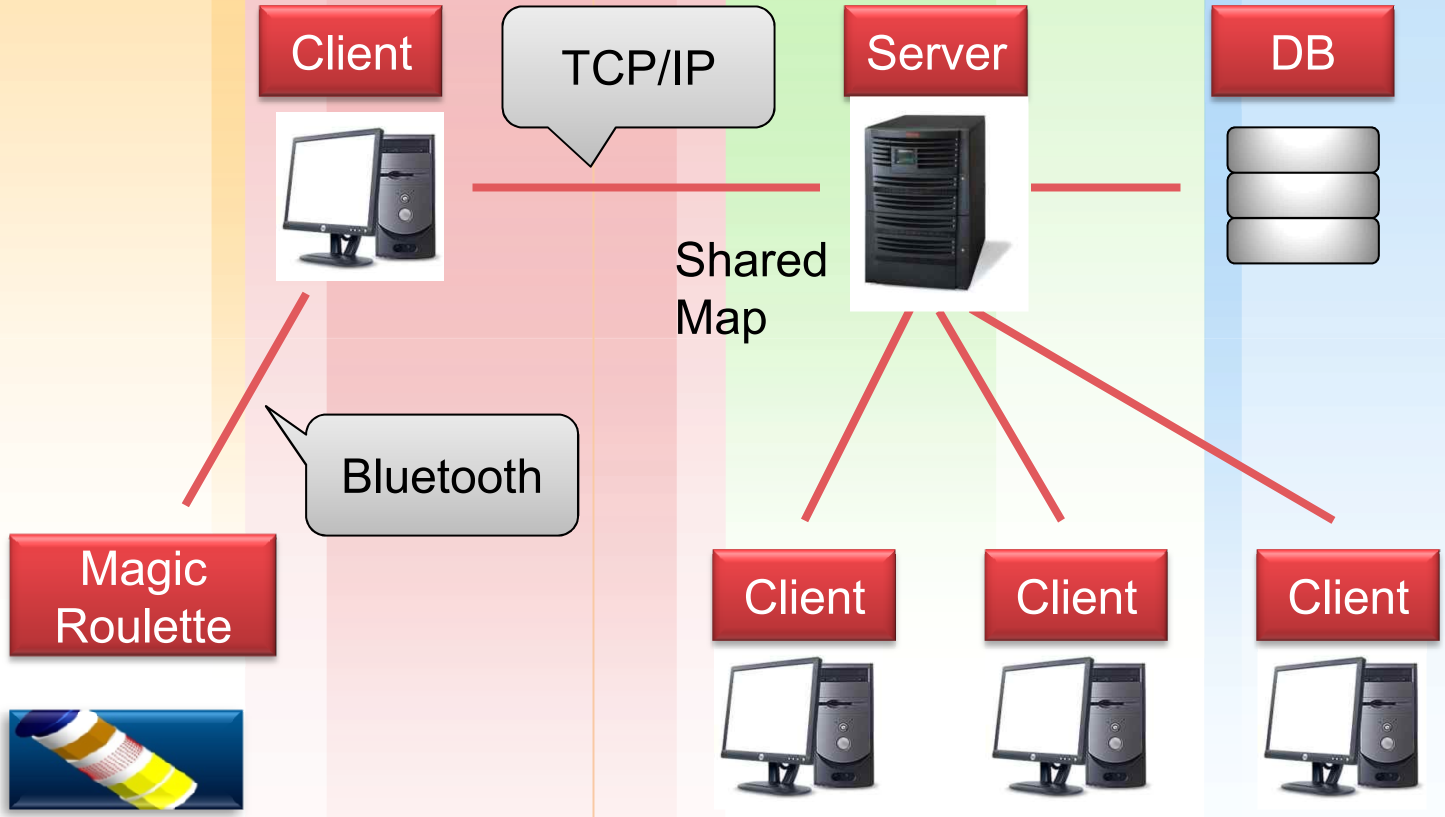


```
#include <stdio.h>

int x;

int main(){
        switch(x){
                case 1:
                        x++;
                        break;
                case 2:
                        x++;
                        break;
                case 3:
                        x++;
                        break;
                default:
                        x++;
                        break;
        }
        while(x<2){
                if(x>3){
                        x++;
```

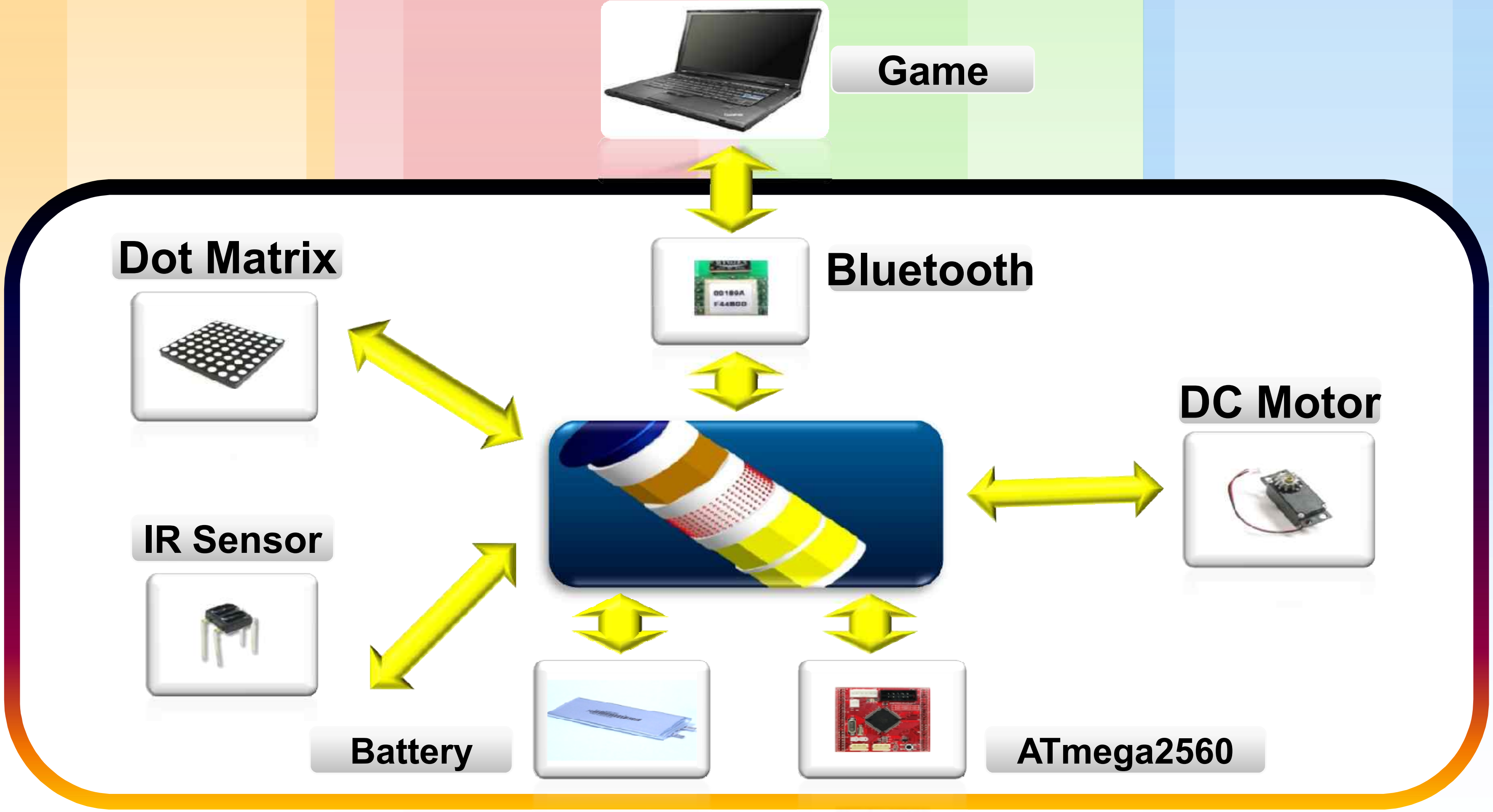
직접작성 x>=5

Architecture

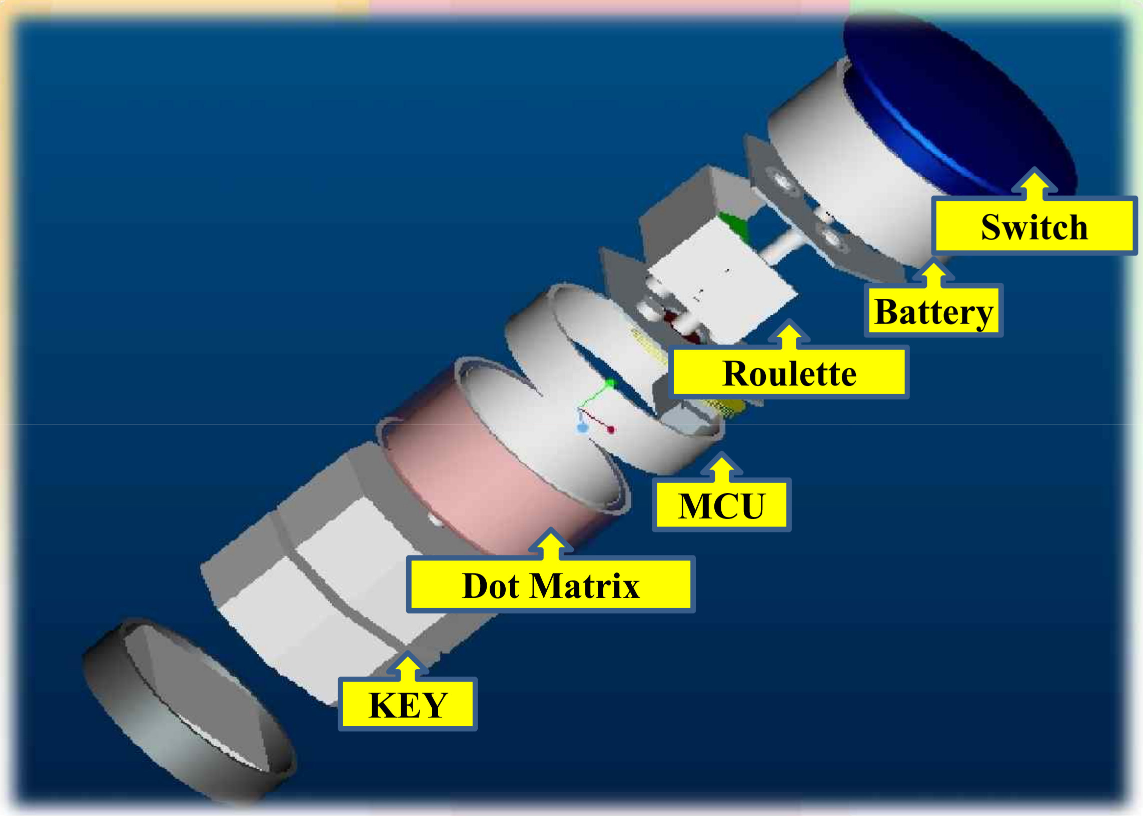
System Architecture

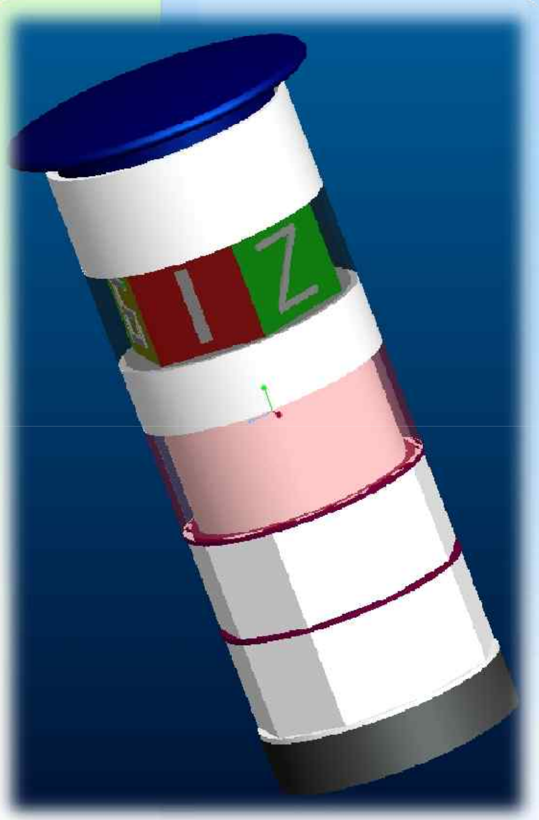


Hardware Architecture(1/3)



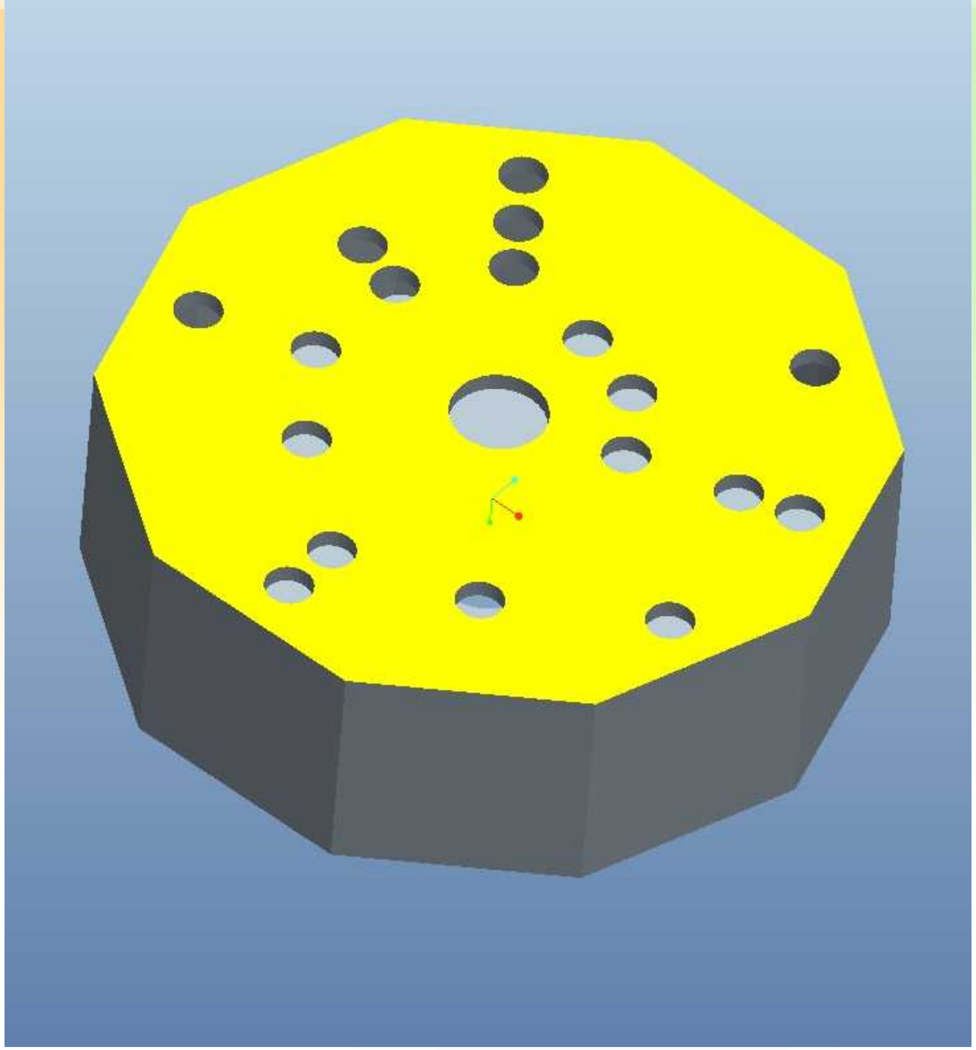
Hardware Architecture(2/3)





Does not interest the typical die & Manipulation, and visualization screen is provided by Roulette Dice!!

Hardware Architecture(3/3)



Contents

x2

- Ball Caster

  Vibration Suppression

- Ball Bearing

  Friction Control

- Maker

  Marker of number

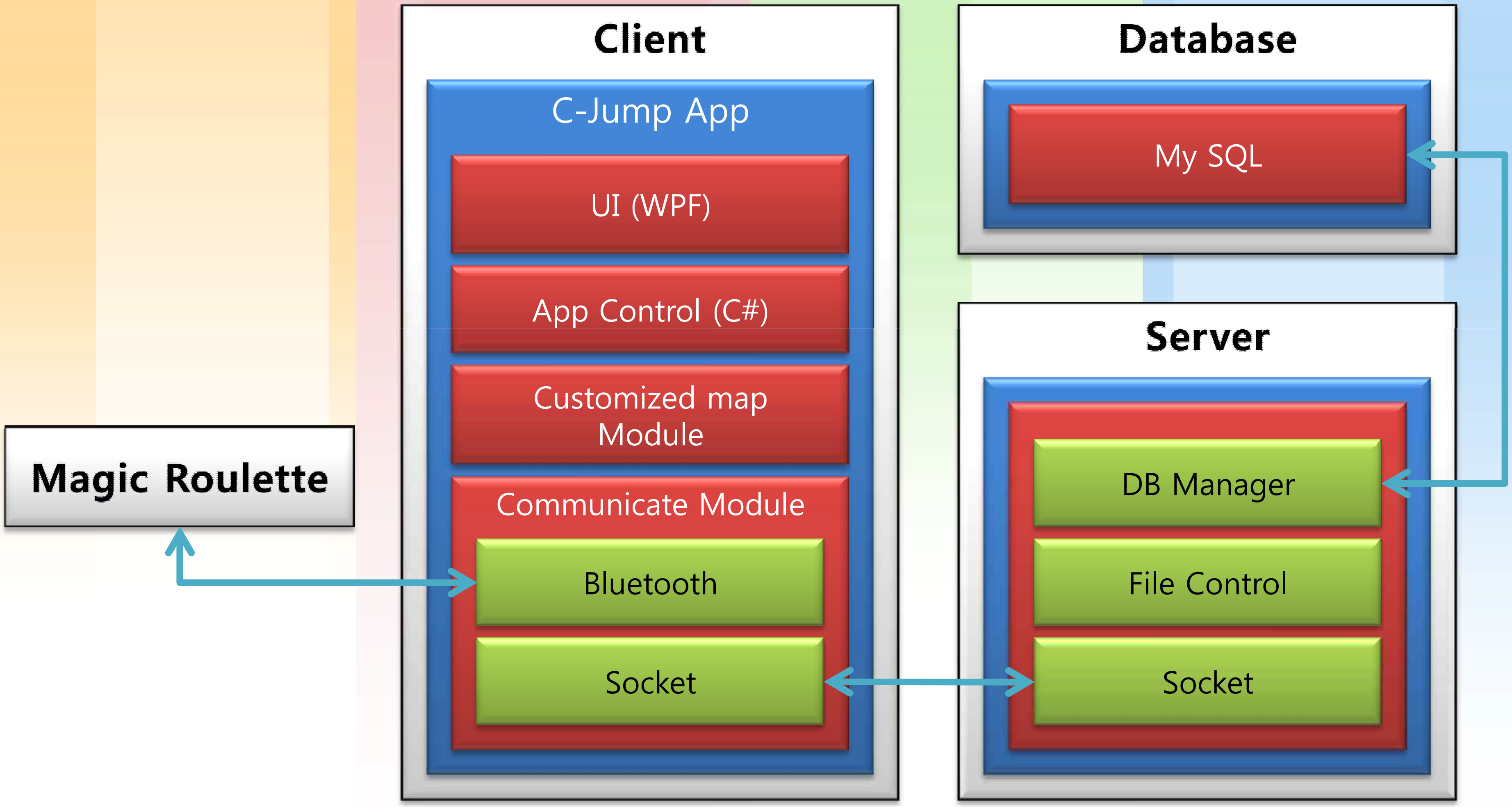
- IR Sensor

  Detection of number

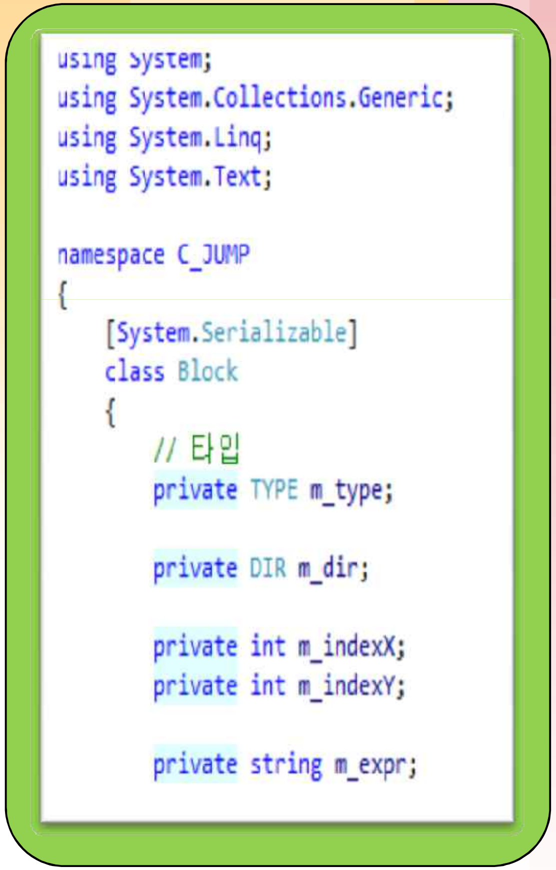
- DC Motor

  Roulette Rotation

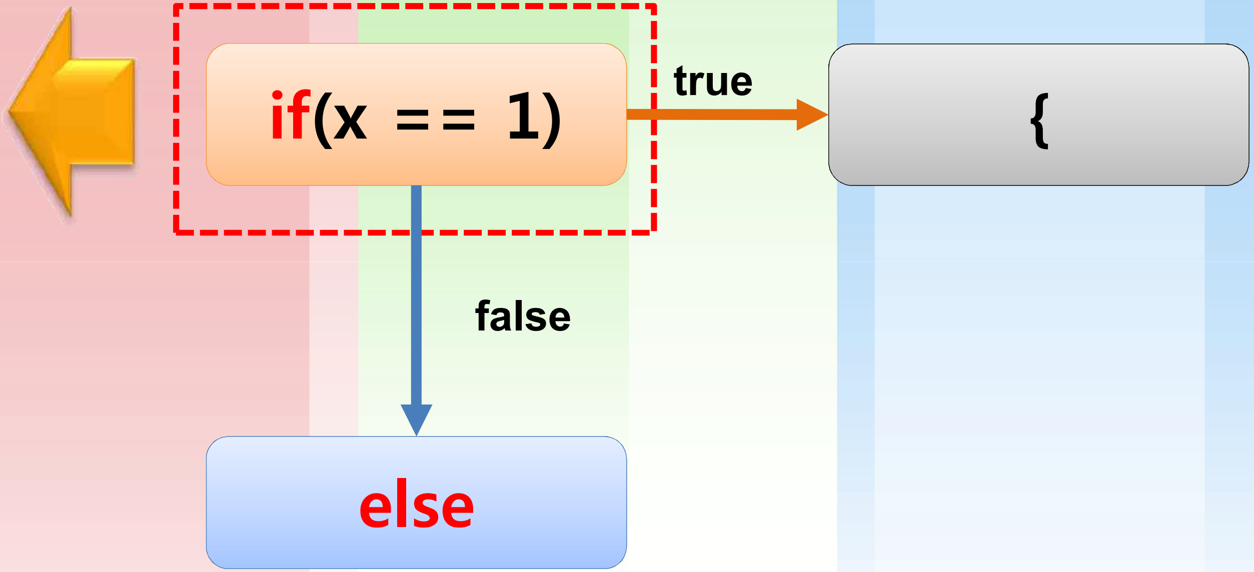
Software Architecture



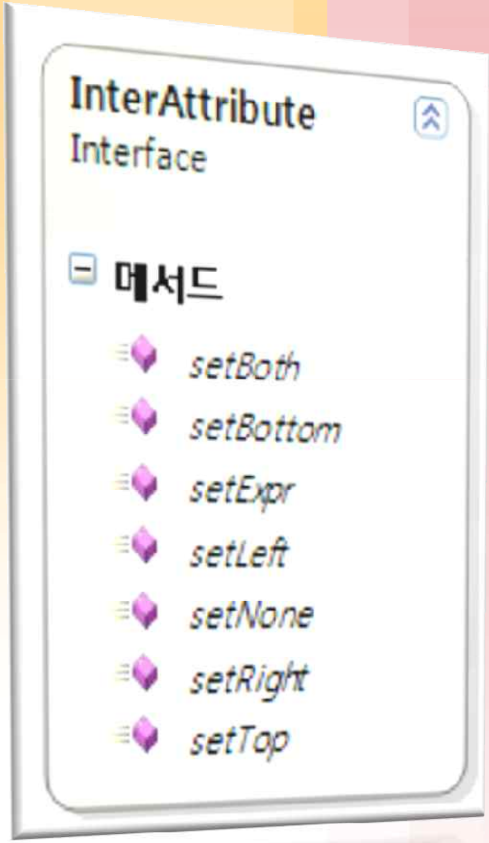
Block Structure



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace C_JUMP
{
    [System.Serializable]
    class Block
    {
        // 타입
        private TYPE m_type;

        private DIR m_dir;

        private int m_indexX;
        private int m_indexY;

        private string m_expr;
```



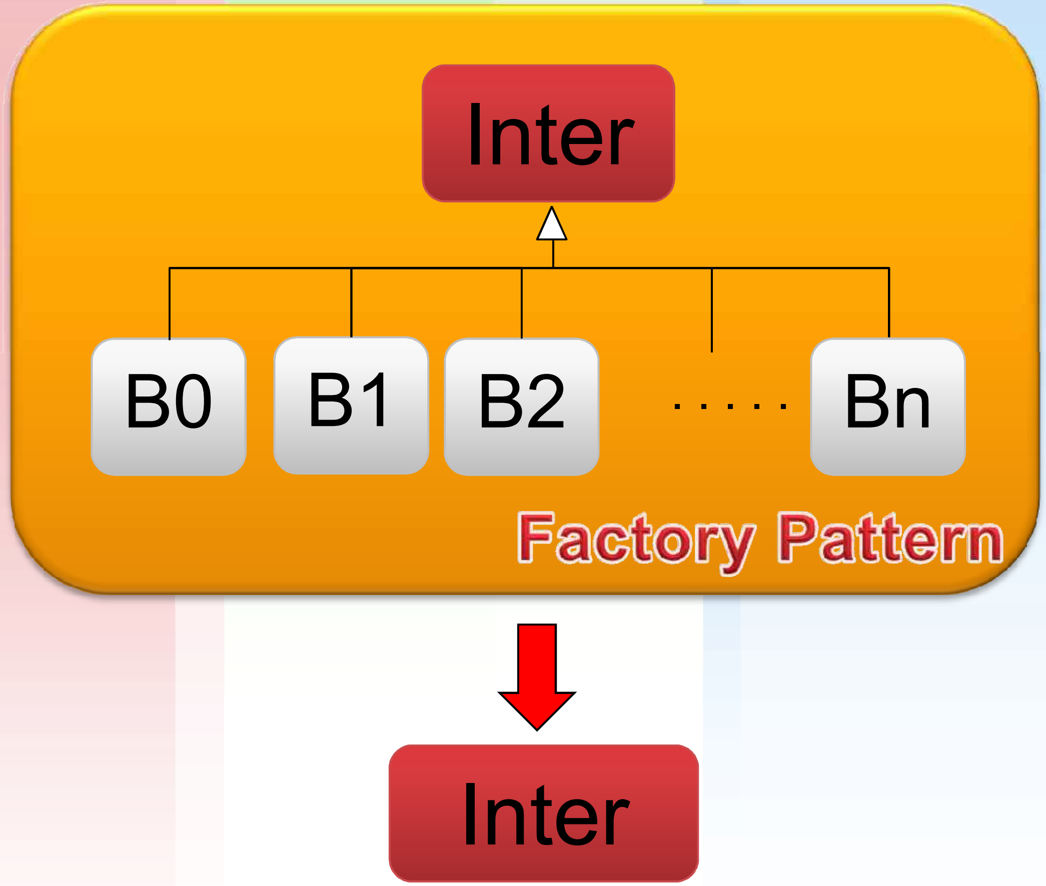
Design Pattern



InterAttribute
Interface

⊟ 메서드

- setBoth
- setBottom
- setExpr
- setLeft
- setNone
- setRight
- setTop



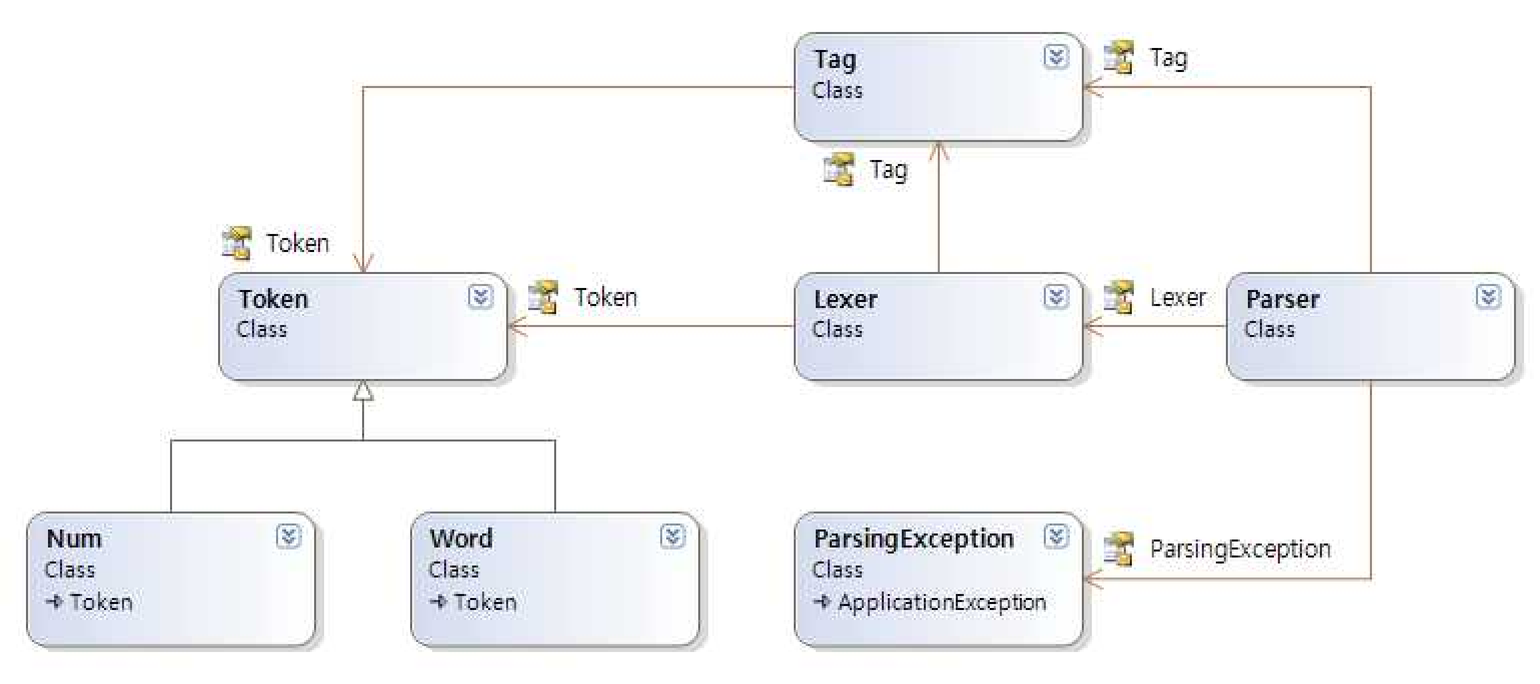
Inter

B0 B1 B2 ...... Bn

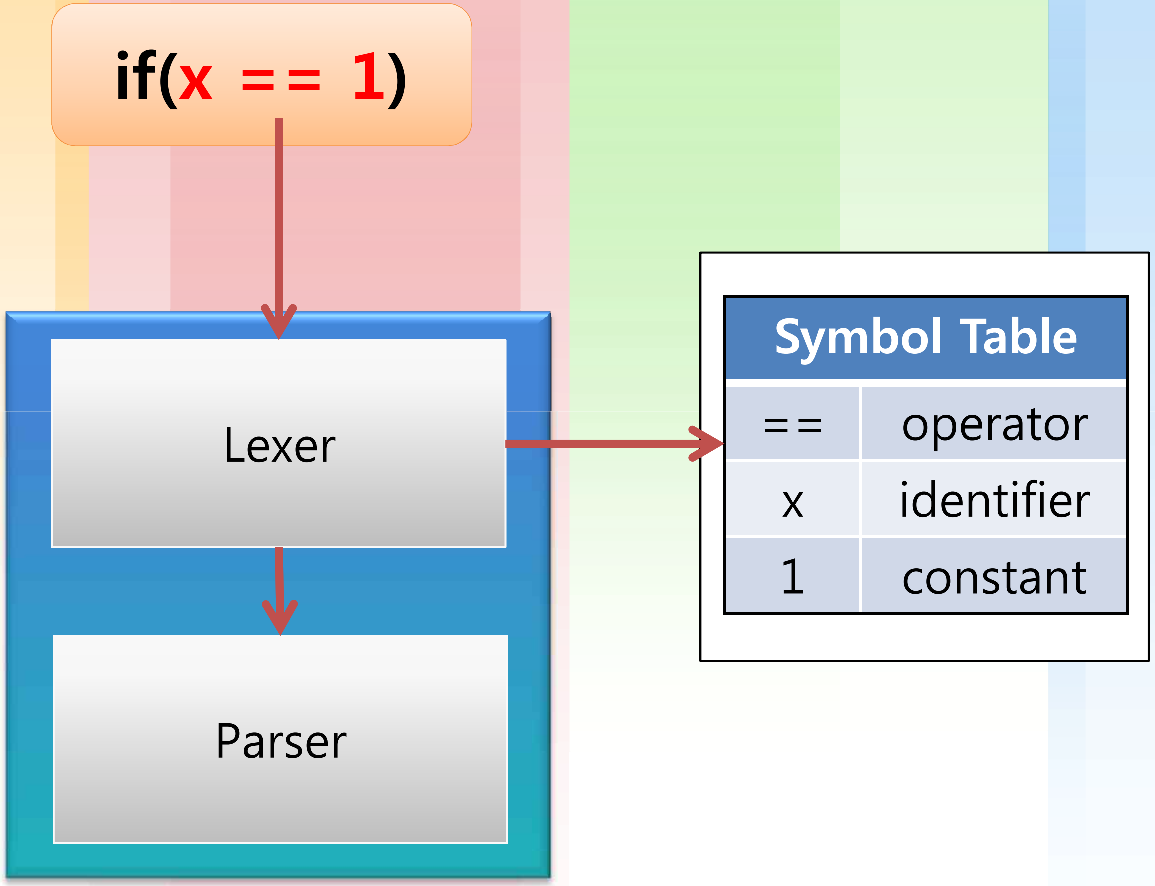
Factory Pattern

Inter

Compiler Structure



Expression Analysis



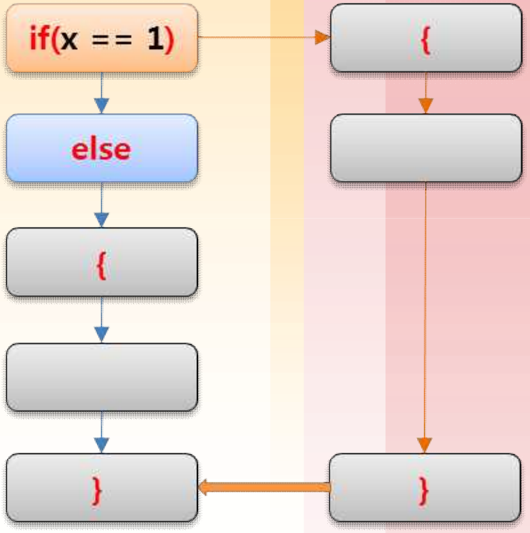
if(x == 1)

Lexer

Parser

Symbol Table	
==	operator
x	identifier
1	constant

Block's Syntax Analysis



if(x == 1) { … }
else { … }

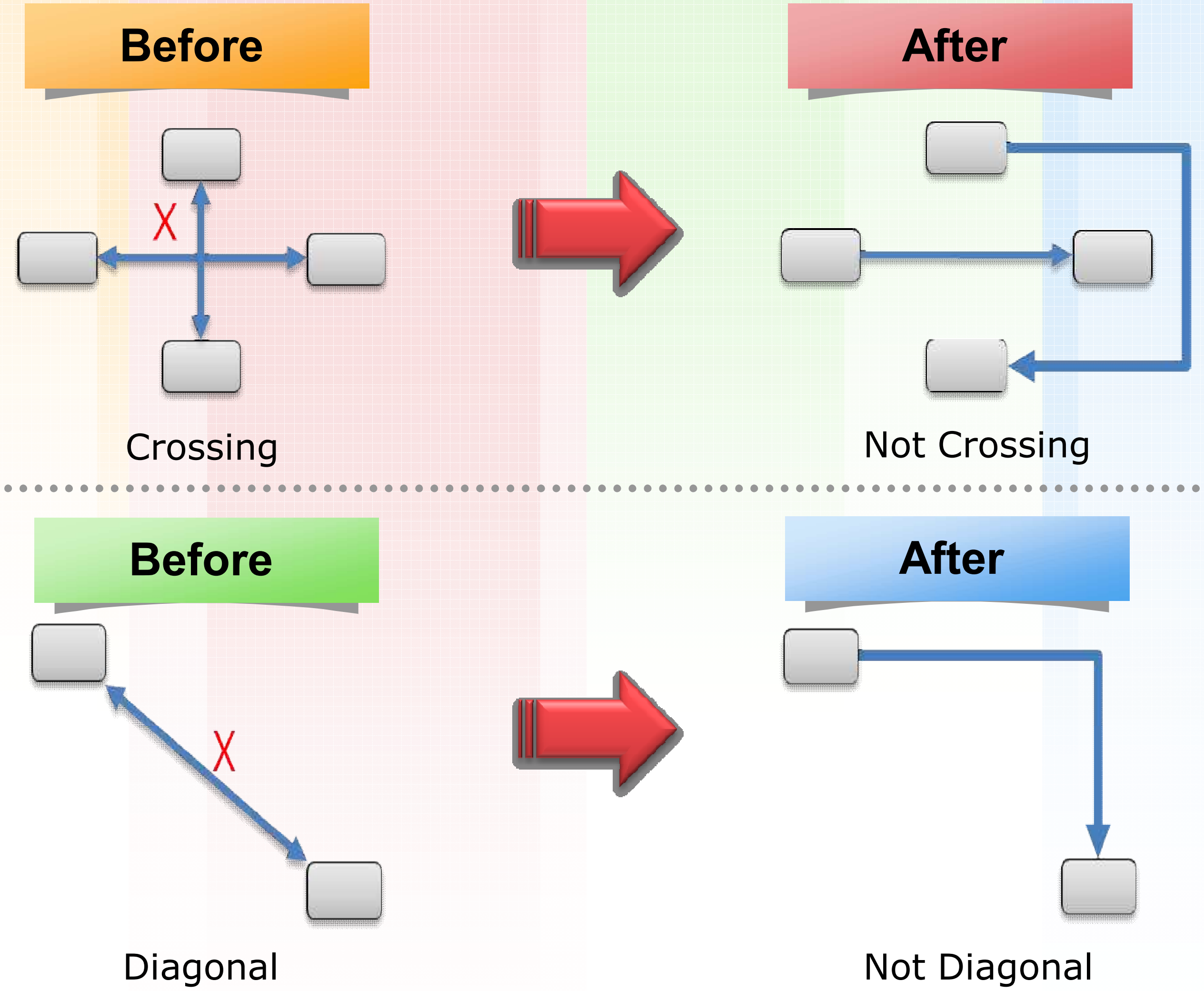
CFG

```
            stmt -> matched_stmt
                  | unmatched_stmt
  matched_stmt -> if expr then matched_stmt else matched_stmt
                  | other
Unmatched_stmt -> if expr then stmt
                  | expr then matched_stmt else unmatched_stmt
```

Compiler CFG

```
<program>                   ->      <block>
<block>                     ->      { <stmts> }
<stmts>                     ->      <stmt> <stmts>     |          ∈
<stmt>                      ->      if ( <equal> ) { <stmts> } else { <stmts> }
                            |       while ( <equal> ) { <stmts> }
                            |       switch ( <equal> ) { <switch_block_stmt_group> ? <switch_labels> ? }
                            |       <equal> ;
                            |       break;
<switch_block_stmt_groups>  ->      <switch_block_stmt_group>
                            |       <switch_block_stmt_groups> <switch_block_stmt_group>
<switch_block_stmt_group>    ->     <switch_labels> <stmts>
<switch_labels>             ->      <switch_label> | <switch_labels> <switch_label>
<switch_label>              ->      case num : <stmts>
                            |       default : <stmts>
                            |       }
<equal>                     ->      <equal> == <rel>
                            |       <equal> != <rel>
                            |       <rel>
<rel>                       ->      <rel> < <expr>
                            |       <rel> <= <expr>
                            |       <rel> > <expr>
                            |       <rel> >= <expr>
                            |       <expr>
<expr>                      ->      <expr> + <term>
                            |       <expr> - <term>
                            |       <term>
<term>                      ->      <term> * <unary>
                            |       <term> / <unary>
                            |       <unary>
<unary>                     ->      <unary> ++ ;
                            |       <unary> -- ;
                            |       <factor>
<factor>          ->        ( <expr> )
                            |       num
                            |       id
```

Move Path



Before

After

Crossing

Not Crossing

Before

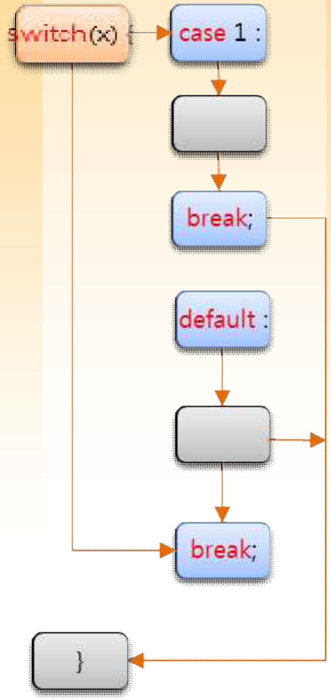
After

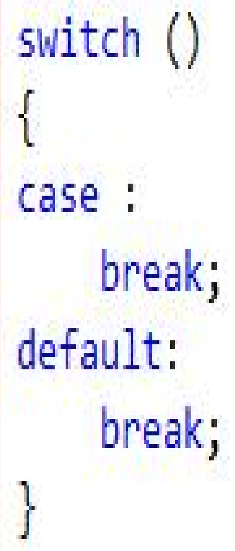
Diagonal

Not Diagonal

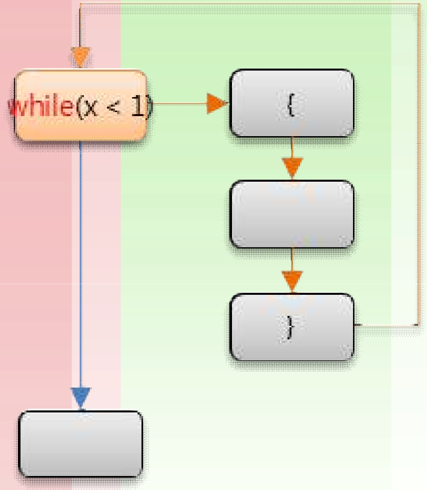
Assist

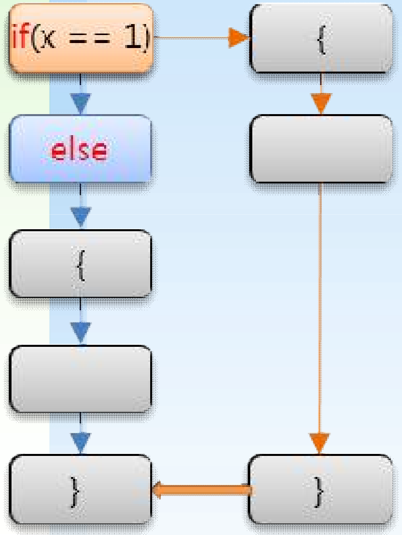
It is easy to use and
Just one click made C-grammar



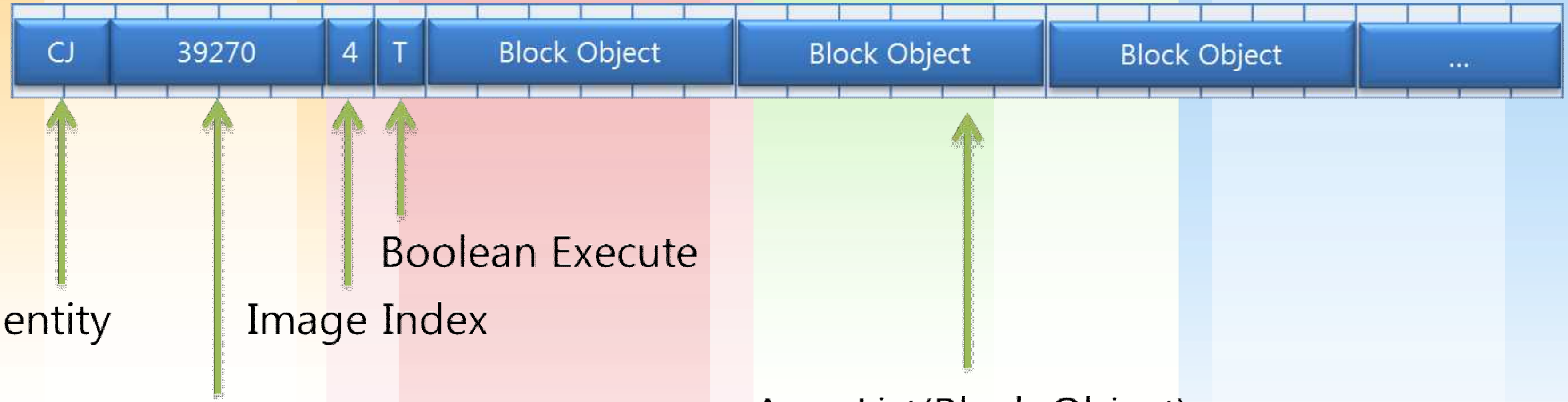


```
switch ()
{
case :
      break;
default:
      break;
}
```





FileFormat



CJ	39270	4	T	Block Object	Block Object	Block Object	...

Boolean Execute

Identity

Image Index

CJ File Size
(Byte)

ArrayList(Block Object)

향후 발전방향

- C JUMP 의 제한적인 C언어 모두 지원
  - 제어문, 함수, 포인터를 통한 연령대별 지원
- Drag & Drop을 이용한 프로그래밍 강화
- Game, Compiler, Map Edit 분리 및 모듈화
- C언어 뿐만이 아닌 Java 까지 발전

Schedule

Kwon Yong Hyun	1 Week							2 Week							3 Week							4 Week							5 Week							6 Week							7 Week							8 Week						
Server Design	🟩	🟩																																																						
Network Protocol			🟩	🟩																																																				
Database					🟩	🟩																																																		
Server Implementation							🟩	🟩	🟩																																															
File Format										🟩	🟩	🟩																																												
Map Up/Down Load(Client)													🟩	🟩	🟩																																									
Grade & Evaluate																🟩	🟩																																							
File Save & Load																		🟩	🟩	🟩	🟩	🟩	🟩																																	
Map Editor Control Make																								🟩	🟩	🟩	🟩	🟩																												
Drag & Drop																													🟩	🟩	🟩	🟩	🟩																							
Magnet Effect																																		🟩	🟩	🟩	🟩	🟩	🟩																	
Undo																																								🟩	🟩	🟩	🟩													
Move Path																																												🟩	🟩	🟩	🟩	🟩	🟩							
Assistant																																																		🟩	🟩	🟩				
Test & Debug																																																					🟩	🟩	🟩	🟩

Jun Sang Gyun	1 Week							2 Week							3 Week							4 Week							5 Week							6 Week							7 Week							8 Week						
Map Editor Design	🟦	🟦																																																						
Block Structure			🟦	🟦	🟦	🟦																																																		
BNF							🟦	🟦	🟦	🟦																																														
Lexical Analysis											🟦	🟦	🟦	🟦	🟦																																									
Syntax Analysis																🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦																								
Bluetooth Protocol																																	🟦	🟦																						
Bluetooth Data Parsing																																			🟦	🟦																				
Block Position Check																																					🟦	🟦	🟦	🟦																
Source Extract																																									🟦	🟦	🟦													
Error Line Check																																												🟦	🟦	🟦										
Block Mapping																																															🟦	🟦	🟦							
Assistant																																																		🟦	🟦	🟦				
Test & Debug																																																					🟦	🟦	🟦	🟦

Kim Sung Han	1 Week							2 Week							3 Week							4 Week							5 Week							6 Week							7 Week							8 Week						
Game Rule	■	■																																																						
UI Control			■	■	■	■	■	■	■																																															
Room Setting														■	■																																									
BGM Sampling																■	■																																							
Sound Management																		■																■	■	■																					
Animation																			■	■										■	■	■	■								■	■									■	■							
Game UI Design											■	■	■														■	■	■									■	■	■							■	■	■										
Turn Control																							■	■	■																															
Map Preview																					■	■																																		
Tutorial Design																																									■	■	■														
Score																																												■												
Trainning Implements																																																			■	■	■				
Test & Debug																																																						■	■	■	■

Kwon Duck Hyun	1 Week							2 Week							3 Week							4 Week							5 Week							6 Week							7 Week							8 Week						
Dot-Matrix Implements	■	■	■	■	■	■	■	■	■																																															
Roulette Manufacture										■	■	■	■	■	■	■	■	■																																						
Key Input Manufacture																			■	■	■	■	■	■	■	■	■	■																												
Safety Circuit																													■	■	■																									
Frame Manufacture																																■	■	■	■	■	■	■																		
Firmware Programing																																							■	■	■	■	■	■	■											
Temporary assembly																																														■	■	■	■							
Test & Debug																																																		■	■	■	■	■	■	■

Q n A

Thank You!