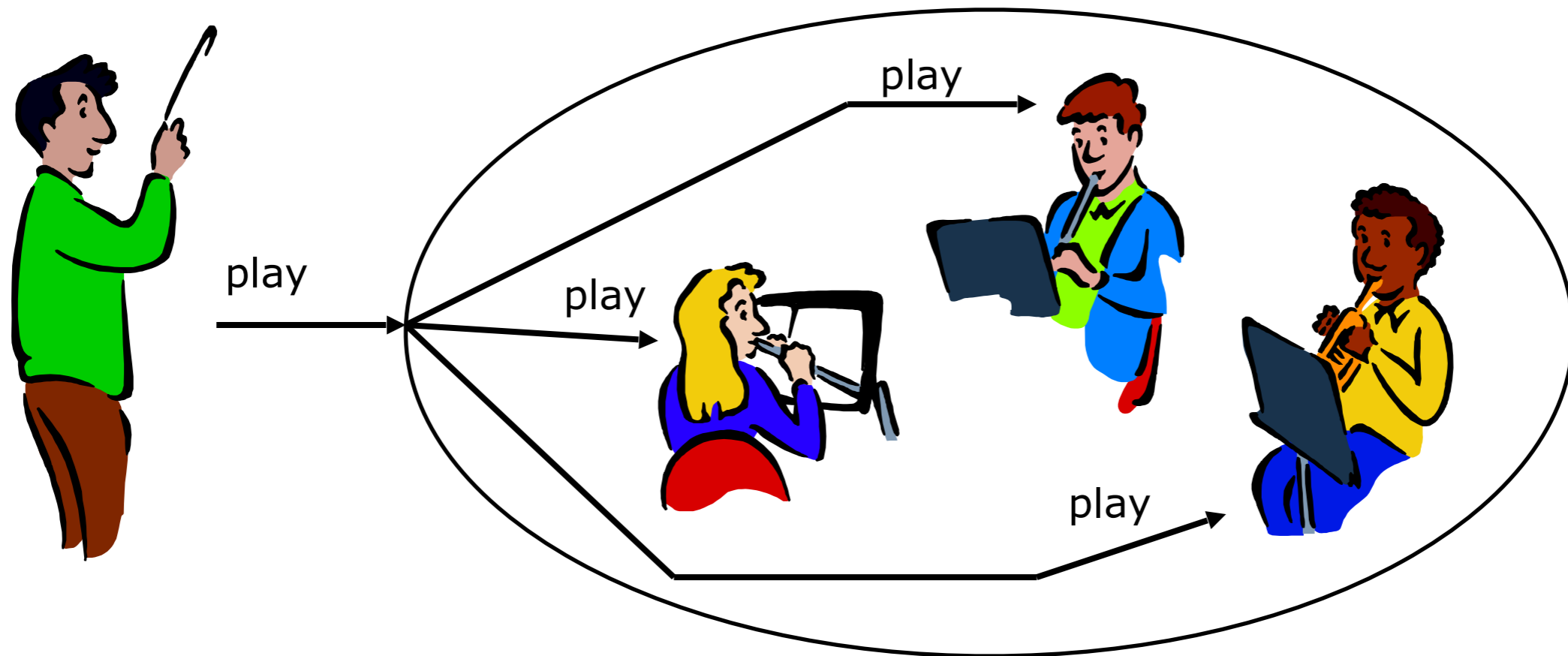


Complexe Interacties



Een optimale werkverdeling



3. Aanpasbaarheid

versie 13 (displayGame)

- 0, 1, infinity principle
- say things once and only once

versie 14 (displayGame)

- coupling / cohesion
- [testen] ASCII uitvoer

versie 15 (Player)

- domeinmodel

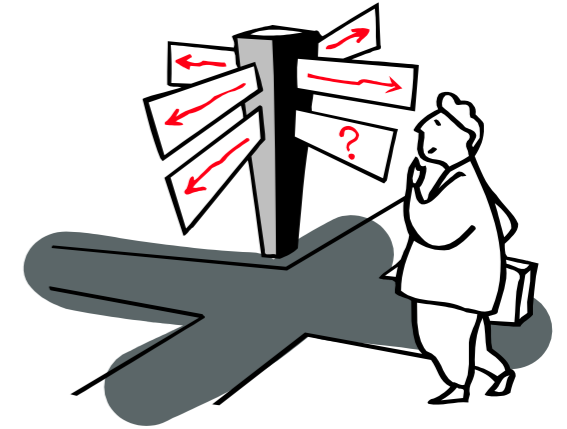
versie 16 (Player.moves)

- vermijd gebruikersinvoer

versie 17 (Player.winner())

- Basisfunctionaliteit

Conclusie



Vereisten



| Vereisten | Technieken |
|--|--|
| <ul style="list-style-type: none">• Betrouwbaarheid• Aanpasbaarheid• Planning | <ul style="list-style-type: none">• Testen + Contracten• Objectgericht ontwerp• Tijdsschatting |
| Werktuigen | |
| <ul style="list-style-type: none">• "Build"• Editor• Debugger• Documentatie | <ul style="list-style-type: none">• Make• Eclipse• Doxygen |

Magic Numbers

```
void displayGame(TicTacToe& ttt) {  
    ...  
    for (row = '1'; row <= '3'; row++) {  
        cout << row;  
        for (col = 'a'; col <= 'c'; col++) {  
            ...  
        }  
    }  
}
```

TicTacToe12

```
void TicTacToe::doMove() {  
    ...  
    col = (char) (_nrOfMoves % 3) + 'a';  
    row = (char) (_nrOfMoves / 3) + '1';  
    ...  
}
```

```
TEST_F(TicTacToeTest, HappyDay) {  
    ...  
    for (col = 'a'; col <= 'c'; col++)  
        for (row = '1'; row <= '3'; row++) {  
            ...  
        }  
}
```

Vuistregel



"0, 1, infinity" principle
Allow - none of foo,
- one of foo, or
- any number of foo.

Waarom ?

zet geen arbitraire limieten

⇒ Vroeg of laat zullen deze limieten aangepast moeten worden

Refactor:

creëer een constante die de arbitraire waarde voorstelt

Vuistregel



say things
once and only once

Variant op
"Keep It Stupidly Simple"

Waarom ?

 duplicatie en redundantie

 ⇒ veranderen moet op twee, drie ... plaatsen

 (eentje wordt vroeg of laat vergeten)

Refactor:

 creëer abstractie via naam, functie, klasse, ...

Y2K

kalenderjaar = laatste
twee cijfers

'98 '99 ?? '00 '01



Y2K Cost in Perspective

- (Sources: Gartner Group and Congressional Research Service)
- World War II: \$4200 billion
- Millennium bug (Y2K): \$600 billion
- Vietnam conflict: \$500 billion
- Kobe earthquake: \$100 billion
- Los Angeles earthquake: \$60 billion

IPv4 address space

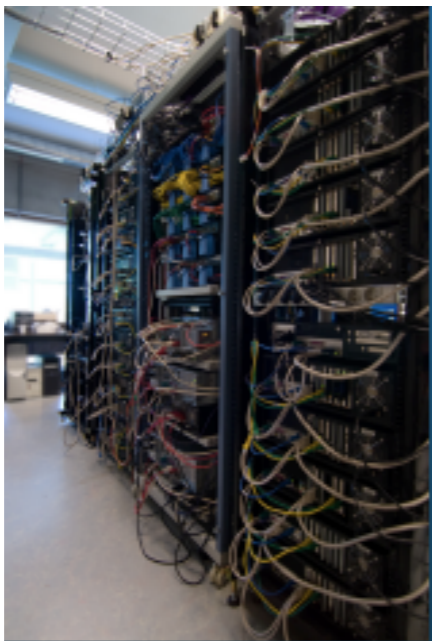
Internet Protocol (version 4)

address ruimte = 2^{32}

$\approx 4 \times 10^9 \approx 4$ miljard

3 februari 2011

- laatste nummer toegewezen door Internet Assigned Numbers Authority (IANA)



Internet Protocol (version 6)

address ruimte = 2^{128}

$\approx 3,4 \times 10^{38}$

∇ aardbewoner

± 50 quadriljard adressen



Magic Numbers \Rightarrow Constantes

TicTacToe13

```
void displayGame(TicTacToe& ttt) {  
    ...  
    for (row = minRow; row <= maxRow; row++) {  
        cout << row;  
        for (col = minCol; col <= maxCol; col++) { ...  
    }  
}
```

```
void TicTacToe::doMove() {  
    ...  
    col = (char) (_nrOfMoves % 3) + minCol;  
    row = (char) (_nrOfMoves / 3) + minRow;  
    ...  
}
```

```
TEST_F(TicTactToeTest, HappyDay) {  
    ...  
    for (col = minCol; col <= maxCol; col++)  
        for (row = minRow; row <= maxRow; row++) {  
    ...  
}
```

Pass by Reference vs. Pass by Value (1/2)

TicTacToe13

```
void displayGame(TicTacToe& ttt) {  
    ...  
}
```

```
TicTacToe numberOfMoves = 1
```

```
  a  b  c  
-----  
1 | 0 |  |  |  
2 |  |  |  |  
3 |  |  |  |  
-----
```

```
TicTacToe numberOfMoves = 2
```

```
  a  b  c  
-----  
1 | 0 | X |  |  
2 |  |  |  |  
3 |  |  |  |  
-----
```

```
void displayGame(TicTacToe ttt) {  
    ...  
}
```

```
TicTacToe.cpp:44: failed assertion  
    `TicTacToe wasn't initialized ...  
TicTacToe numberOfMoves = Abort trap: 6
```



Pass by Reference vs. Pass by Value (2/2)

properlyInitialized \Rightarrow defensieve programmeerstijl

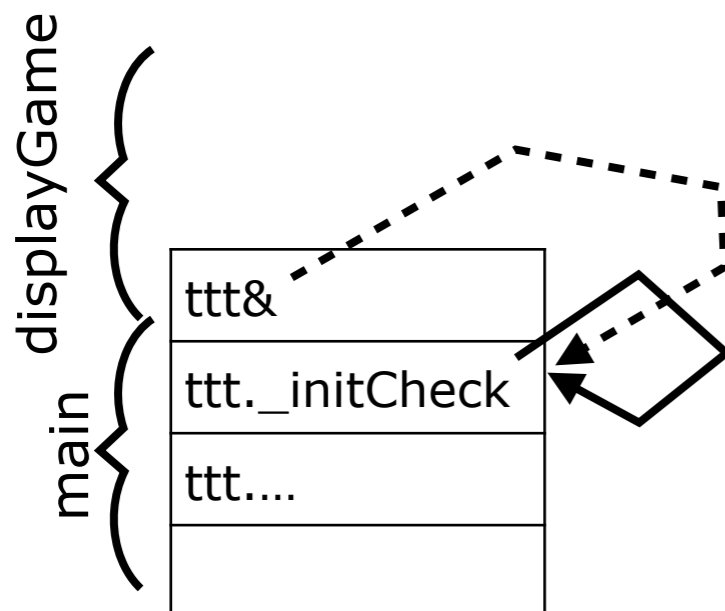
```
void displayGame(TicTacToe& ttt) {
```

...

```
int main(...) {  
    TicTacToe ttt;
```

...

Pass by Reference



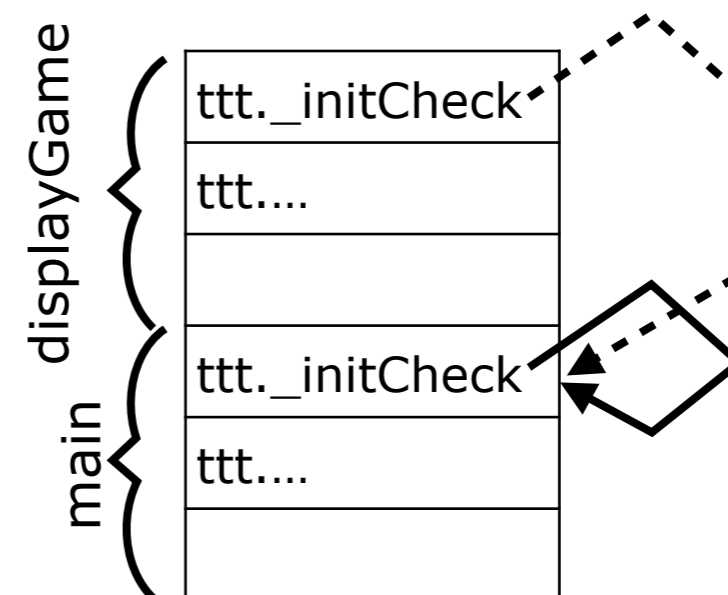
```
void displayGame(TicTacToe ttt) {
```

...

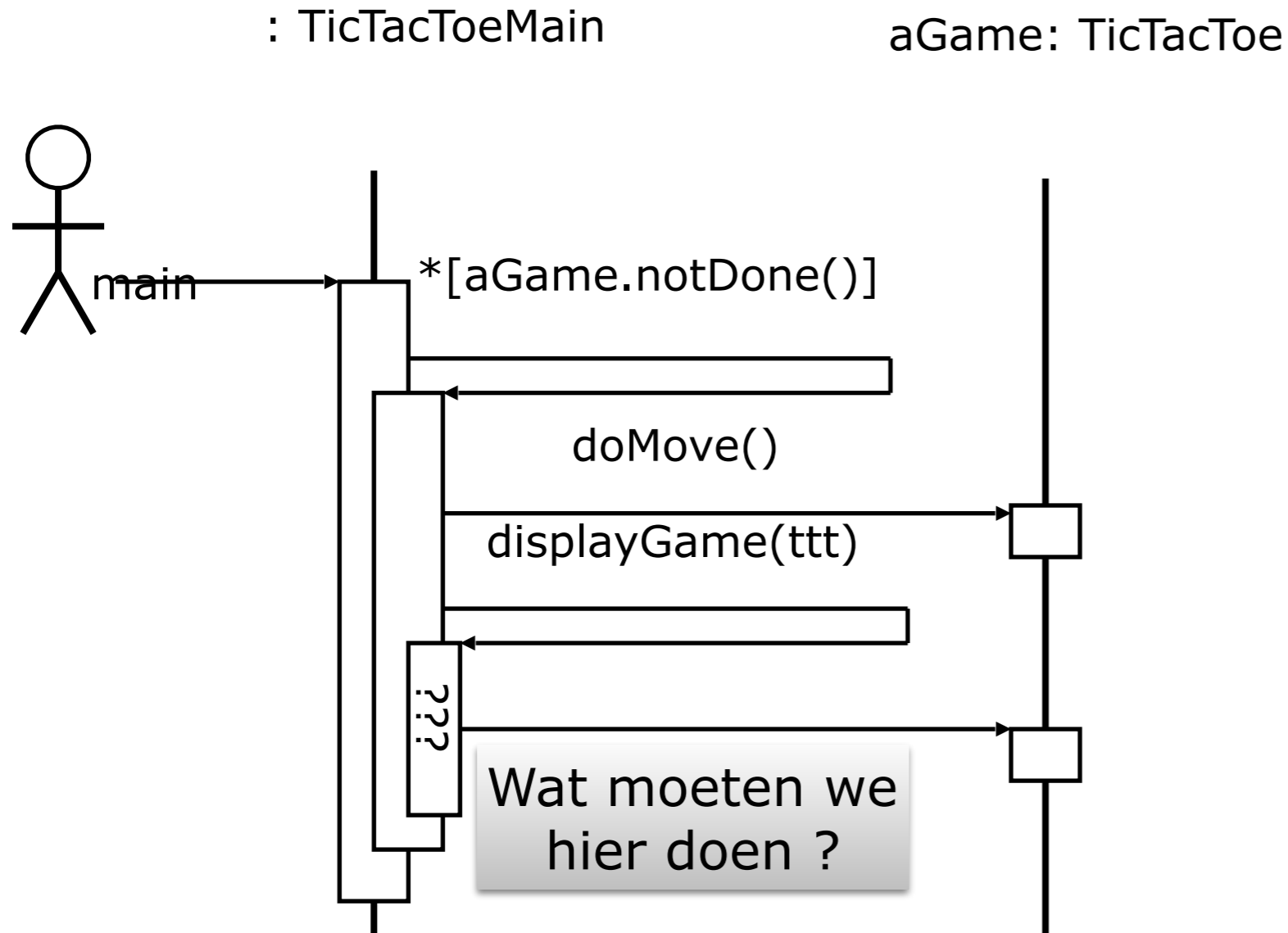
```
int main(...) {  
    TicTacToe ttt;
```

...

Pass by Value



TicTacToe13



```
int main(int argc, char **argv) {
    TicTacToe ttt;

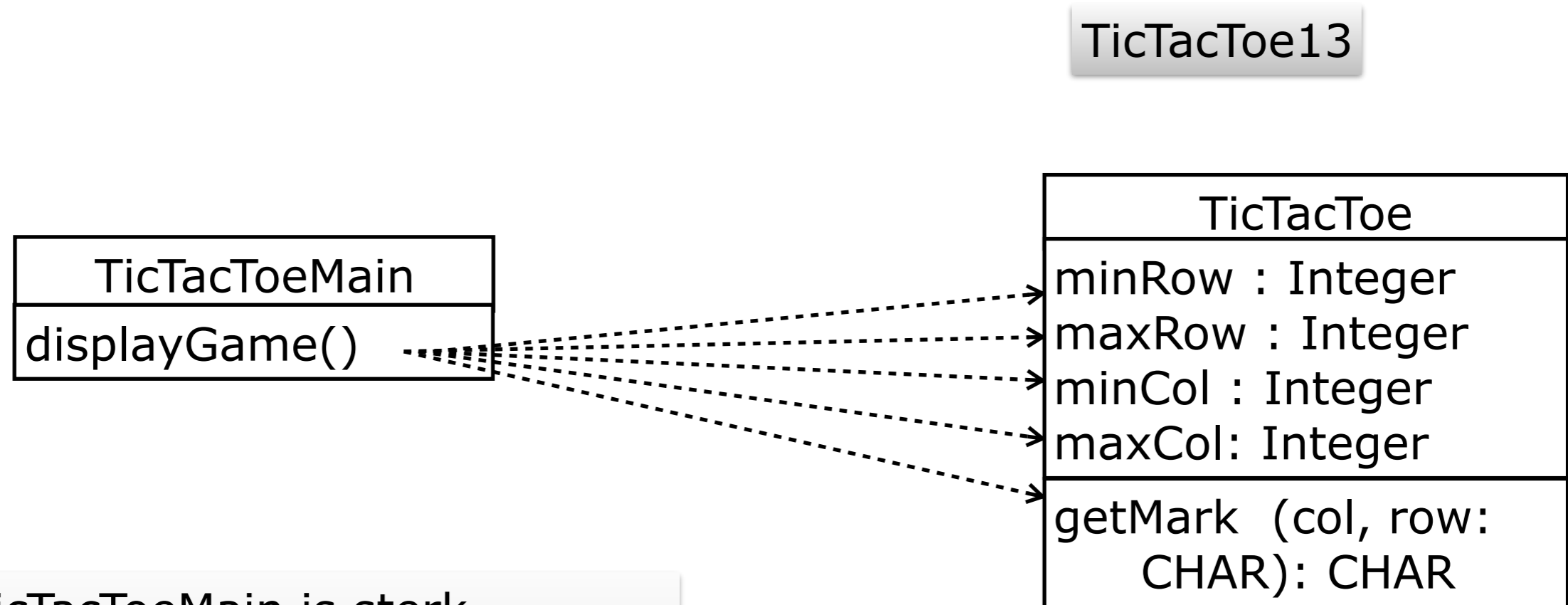
    while (ttt.notDone()) {
        ttt.doMove();
        cout << endl << endl;
        displayGame(ttt);
    };
}
```

Sterke Koppeling (code)

```
void displayGame(TicTacToe& ttt) {
    char col, row;
    cout << "TicTacToe numberOfMoves = " << ttt.nrOfMoves() << endl;
    cout << "    a    b    c    " << endl;
    cout << "    ----- " << endl;
    for (row = minRow; row <= maxRow; row++) {
        cout << row;
        for (col = minCol; col <= maxCol; col++) {
            cout << " | " << ttt.getMark(col, row);
        }
        cout << " |" << endl;
    };
    cout << "    ----- " << endl;
}
```

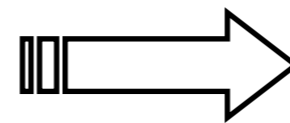
Afhankelijk van
4 constanten &
1 operatie

Sterke Koppeling



TicTacToeMain is sterk gekoppeld aan TicTacToe

- Er zijn veel afhankelijkheden van TicTacToeMain naar TicTacToe
- Een verandering aan TicTacToe betekent meestal dat we ook TicTacToeMain moeten aanpassen



Aanpasbaarheid :(

Vuistregel



Plaats het gedrag
dicht bij de data

Waarom ?

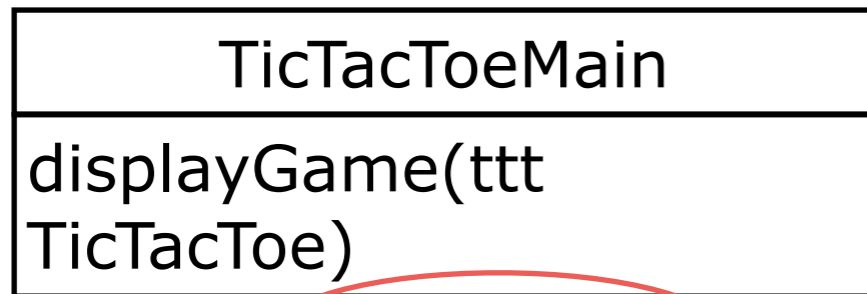
- veranderingen zullen eerder lokaal effect hebben
⇒ zwakke koppeling

Refactor:

- Routines die veel "get" operaties oproepen
⇒ verplaats naar de corresponderende klasse.

Zwakke Koppeling

TicTacToe13



```
void displayGame(TicTacToe& ttt) {  
    ...  
    for (row = minRow;  
        row <= maxRow; row++) {  
        ... ttt.getMark(col, row);  
    }  
}
```

TicTacToeMain is *zwak* gekoppeld aan TicTacToe

- Er is maar één afhankelijkheid van TicTacToeMain naar TicTacToe
- Een verandering aan TicTacToe heeft zelden een effect op TicTacToeMain



TicTacToe14



```
void TicTacToe::displayGame() {  
    ...  
    for (row = minRow;  
        row <= maxRow; row++) {  
        ... this->getMark(col, row);  
    }  
}
```

Aanpasbaarheid :)

Koppeling via cout

```
void TicTacToe::displayGame() {
    char col, row;
    REQUIRE(this->properlyInitialized(),
        "TicTacToe wasn't initialized when calling displayGame");
    std::cout << "TicTacToe numberOfMoves = " << this->nrOfMoves()
        << std::endl;
    std::cout << "    a    b    c    " << std::endl;
    std::cout << "    ----- " << std::endl;
    for (row = minRow; row <= maxRow; row++) {
        std::cout << row;
        for (col = minCol; col <= maxCol; col++) {
            std::cout << " | " << this->getMark(col, row);
        }
        std::cout << " |" << std::endl;
    };
    std::cout << "    ----- " << std::endl;
}
```

TicTacToe is afhankelijk van `cout`
⇒ alleen uitvoer op console 😞

Vuistregel



Nooit user-interface code
in de basisklassen

Waarom ?

- Minder kans op veranderingen in basisklassen (user-interface wordt vaak veranderd)

Refactor:

- Extra parameter als in/uitvoerkanal naar de buitenwereld

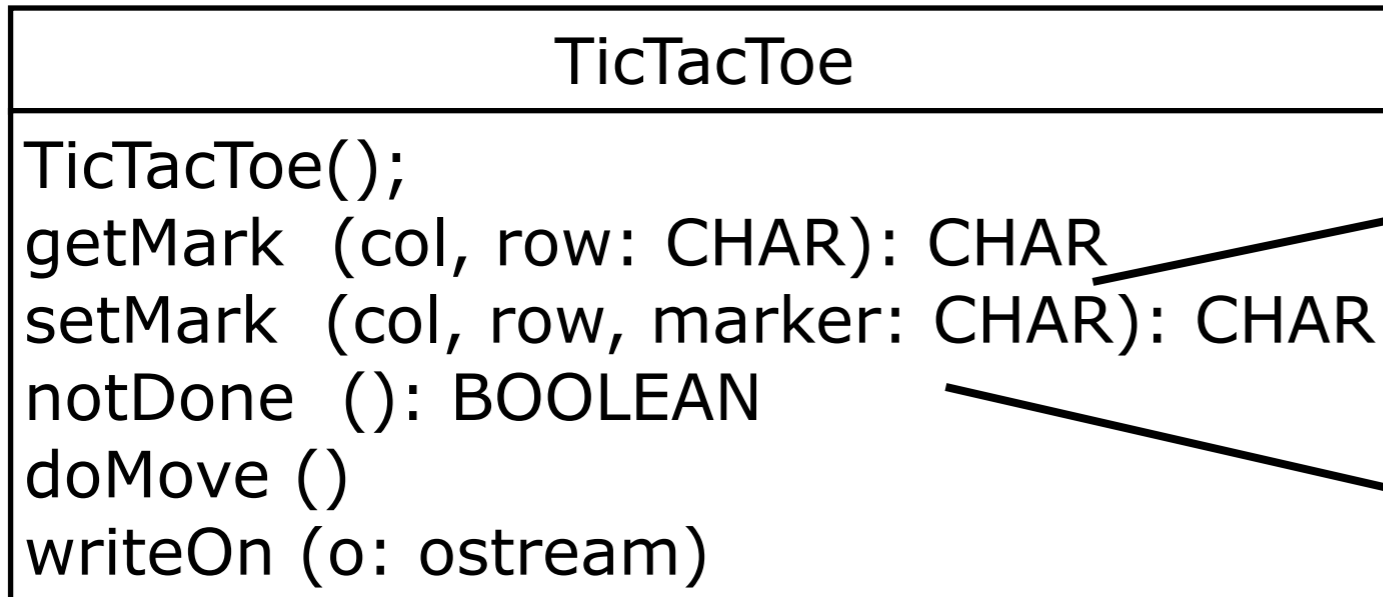
Koppeling via ostream

```
void TicTacToe::writeOn(std::ostream& onStream) {
    char col, row;
    REQUIRE(this->properlyInitialized(),
            "TicTacToe wasn't initialized when calling displayGame");
    onStream << "TicTacToe numberOfMoves = " << this->nrOfMoves() <<
std::endl;
    onStream << "    a    b    c    " << std::endl;
    onStream << "    ----- " << std::endl;
    for (row = minRow; row <= maxRow; row++) {
        onStream << row;
        for (col = minCol; col <= maxCol; col++) {
            onStream << " | " << this->getMark(col, row);
        }
        onStream << " |" << std::endl;
    };
    onStream << "    ----- " << std::endl;
}
```

Extra parameter als uitvoerkanaal
+ betere naamgeving

Cohesie

TicTacToe14



```
<<precondition>>  
properlyInitialized() &  
("a"<=col) & (col<="c") &  
("1"<=row) &  
(row<="3")  
  
<<postcondition>>  
(result = "O") OR (result  
= "X")  
OR (result = " ")
```

TicTacToe is redelijk cohesief

- 1 operatie gebruiken impliceert ook het gebruik van alle andere
 - alle operaties zijn nodig/nuttig
- ⇒ veranderingen aan de interface zijn weinig waarschijnlijk

```
<<precondition>>  
...  
  
<<postcondition>>  
getMark(col, row) =  
marker
```

Aanpasbaarheid 😊

Koppeling vs. Cohesie

Koppeling

= mate waarin een component afhankelijk is van andere componenten

te MINIMALISEREN

⇒ veranderingen hebben minder impact

Cohesie

= mate waarin de onderdelen van een component afhankelijk zijn van elkaar

te MAXIMALISEREN

⇒ veranderingen zijn minder waarschijnlijk

?? Ideaal ??


= een component die niks doet
⇒ perfectie is niet haalbaar

Testen van ASCII uitvoer (TicTacToe14)

```
TicTacToe
TicTacToe();
getMark (col, row: CHAR): CHAR
setMark (col, row, marker: CHAR): CHAR
notDone (): BOOLEAN
doMove ()
writeOn (o: ostream)
```



Hoe testen we de uitvoer ?



Als de functionaliteit groeit,
dan groeit de test mee

Vuistregel



Test lange uitvoer door
het vergelijken van files.

Waarom ?

- Veel & frequent testen \Rightarrow makkelijk om steeds uitvoer te testen

Hoe ?

- Verwachte uitvoerdata wordt één maal manueel gecreëerd
- vergelijk verwachte uitvoer met gegenereerde uitvoer

Testen van uitvoer

```
TEST_F(TicTacToeTest, OutputHappyDay) {  
    ASSERT_TRUE(DirectoryExists("testOutput"));  
    //if directory doesn't exist then no need in proceeding with the test  
  
    ofstream myfile;  
    myfile.open("testOutput/happyDayOut.txt");  
    while (ttt_.notDone()) {  
        ttt_.doMove();  
        ttt_.writeOn(myfile);  
    };  
    myfile.close();  
    EXPECT_TRUE(  
        FileCompare("testOutput/happyDayExpectedOut.txt",  
                   "testOutput/happyDayOut.txt"));  
}
```

Schrijf alles op een bestand ...

... en vergelijk met wat je verwacht

code voor "DirectoryExists" en "FileCompare" ⇒ Zie TicTacToe14



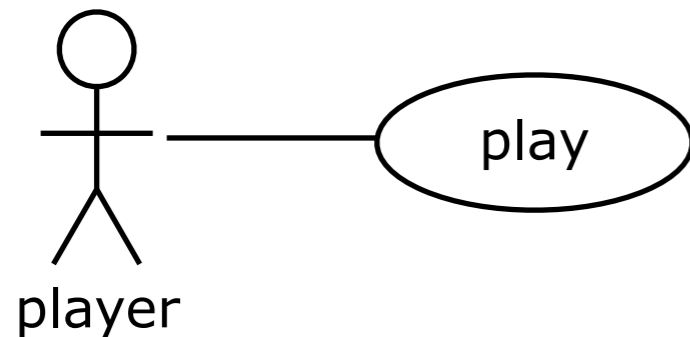
Testing of FileCompare

```
TEST_F(TicTactToeTest, FileCompare) {
    ASSERT_TRUE(DirectoryExists("testOutput"));
    ...
    //compare 2 empty files
    EXPECT_TRUE(FileCompare("testOutput/file1.txt", "testOutput/file2.txt"));
    ...
    //compare an empty and a non-empty files
    ...
    //compare two equal files
    ...
    //compare 2 non-empty files which are off by a character in the middle
    ...
    //compare 2 non-empty files where one is one character shorter than the
other
    ...
    //compare existing against non existing file
}
```

Test the hulpfuncties in de tests !



TicTacToe15



Waar voegen
we dit bij ?

Use Case 1: play

- Goal: 2 players play TicTacToe, 1 should win
- Precondition: An empty 3x3 board
- Success end: 1 player is the winner

Steps

1. Two players start up a game (First is "O"; other is "X")
2. WHILE game not done
 - 2.1 Current player makes move
 - 2.2 Switch current player
3. Announce winner

Vuistregel

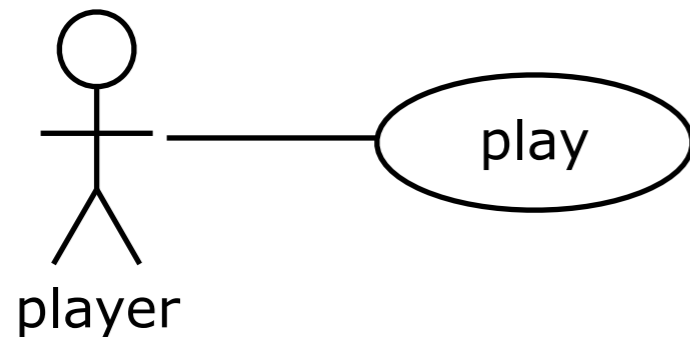


Maak een model van
het probleemdomein
= het DOMEINMODEL

Tips:

- Zelfstandige naamwoord als indicator voor object / klasse.
- Werkwoord als indicator voor operatie
- Naamgeving in basisklassen = naamgeving in probleemdomein

Domeinmodel



Legende

- Substantief
- *Werkwoord*

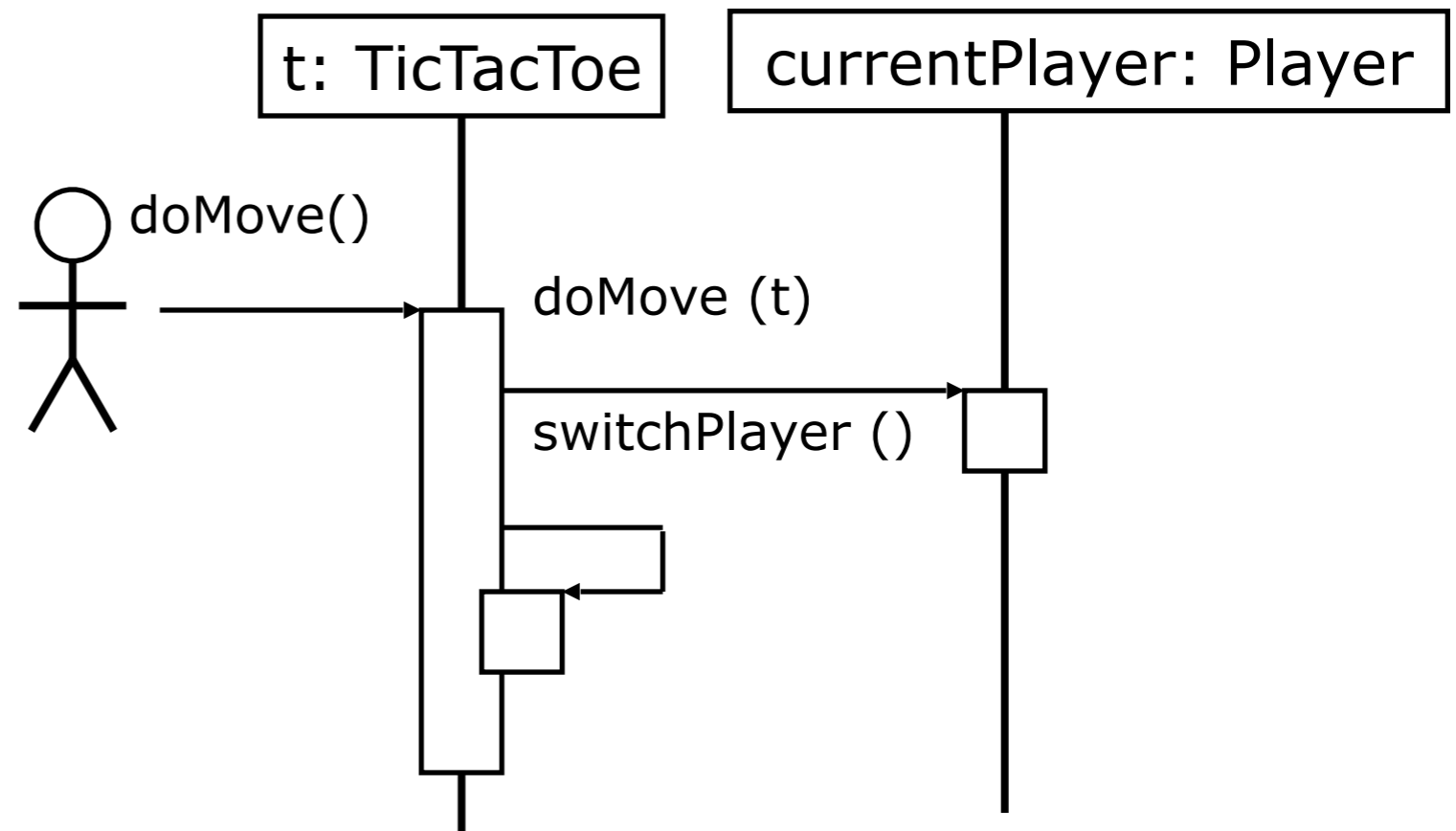
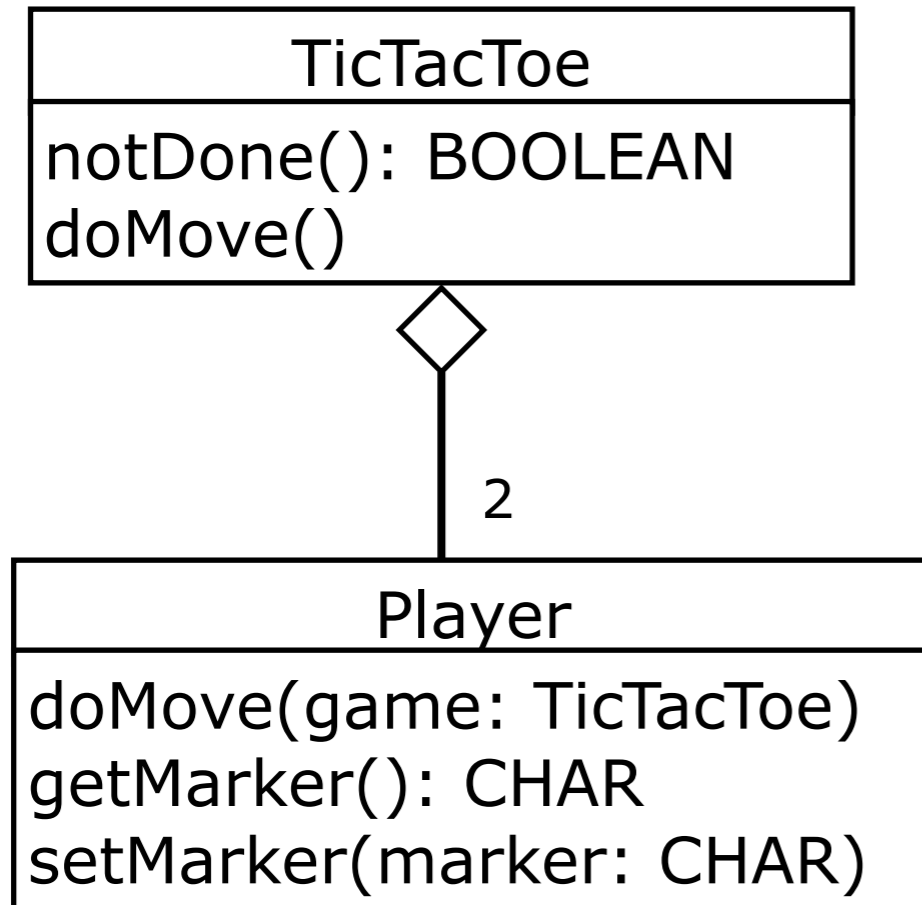
Use Case 1: play

- Goal: 2 players play TicTacToe, 1 should win
- Precondition: An empty 3x3 board
- Success end: 1 player is the winner

Steps

1. Two players *start up* a game (First is "O"; other is "X")
2. WHILE game not done
 - 2.1 Current player *makes move*
 - 2.2 *Switch* current player
3. *Announce* winner

TTT15: Player



Forward Declaration

```
class TicTacToe; // forward declaration
```

```
class TicTacToePlayer {  
    ...  
    void doMove(TicTacToe& game);  
    ...  
private:  
    TicTacToePlayer * _initCheck; //use pointer to myself ...  
    char _marker;  
};
```

```
class TicTacToe {  
    ...  
private:  
    ...  
    TicTacToePlayer _players [2];  
};
```

TicTacToe & TicTacToePlayer
circular afhankelijk !
⇒ forward declaration



Opgelet: Magic Number

Tests blijven lopen ...

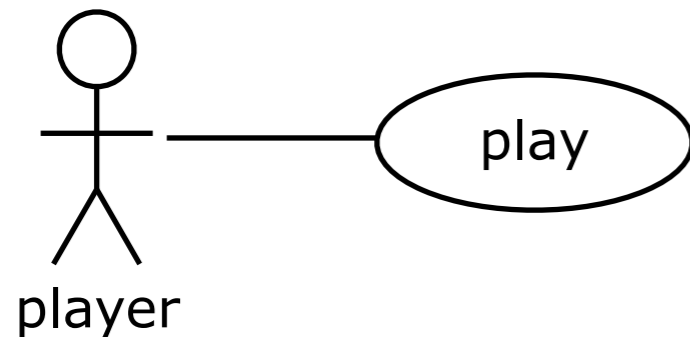
```
void TicTacToePlayer::doMove(TicTacToe& game) {  
    REQUIRE(this->properlyInitialized(),  
            "TicTacToePlayer wasn't initialized when calling getMarker");  
    REQUIRE(game.properlyInitialized(),  
            "game wasn't initialized when passed to Player->doMove");  
    char col, row;  
    col = (char) (game.nrOfMoves() % 3) + minCol;  
    row = (char) (game.nrOfMoves() / 3) + minRow;  
    game.setMark(col, row, _marker);  
}
```



Opgelet: Magic Number

Verplaatst van TicTacToe naar Player
Tests blijven lopen 😊

TicTacToe16



Hoe verkrijgen we gebruikersinvoer ?

Use Case 1: play

- Goal: 2 players play TicTacToe, 1 should win
- Precondition: An empty 3x3 board
- Success end: 1 player is the winner

Steps

1. Two players start up a game (First is "O"; other is "X")
2. WHILE game not done
 - 2.1 Current player makes move
 - 2.2 Switch current player
3. Announce winner

Vuistregel



Een test verwacht géén
gebruikersinvoer

Waarom ?

- Veel & frequent testen \Rightarrow onmogelijk om steeds invoer te geven

Hoe dan wel ?

- Invoerdata wordt gecreëerd door testprogramma (testbestand ?)
- Ontwerp basisklassen onafhankelijk van het data invoer kanaal

TTT17: Reeks van zetten

```
int main(int argc, char **argv) {
    TicTacToe ttt;
    ttt.setMoves("a1c1b2a3c3", "b1a2c2b3");

    while (ttt.notDone()) {
        ttt.doMove();
        cout << endl << endl;
        ttt.writeOn(cout);
    };
}
```

| | a | b | c |
|---|---|---|---|
| 1 | O | X | O |
| 2 | X | O | X |
| 3 | O | X | O |

een reeks zetten
⇒ invoerdata door (test)programma

...

```
void TicTacToe::setMoves(const string oMoves, const string xMoves) {
    // REQUIRE(this->properlyInitialized(), "TicTacToe ...
    // REQUIRE(legalMoves(oMoves), "TicTacToe::setMoves requires ...
    // REQUIRE(legalMoves(xMoves), "TicTacToe::setMoves requires ...
}
```

Geen gebruikersinvoer in Tests

```
// Tests the "happy day" scenario
TEST_F(TicTactToeTest, HappyDay) {
    EXPECT_EQ(0, ttt_.nrOfMoves());
    EXPECT_TRUE(ttt_.notDone());
    ttt_.setMoves("a1c1b2a3c3", "b1a2c2b3");
    while (ttt_.notDone()) {
        ttt_.doMove();
    };
    char col, row;
    bool markIsX = false; // start with 'O'
    for (col = minCol; col <= maxCol; col++)
        for (row = minRow; row <= maxRow; row++) {
            if (markIsX)
                EXPECT_EQ('X', ttt_.getMark(col, row));
            ...
        }
}
```

Geen gebruikersinvoer

Vuistregels (vorige week)

Minstens één test
per "normaal"
scenario



Als de functionaliteit
groeit, dan groeit de
test mee

Waarom ?

Minimum veiligheidsnorm

Hoe ?

Controleer de tussentijdse
toestand via "EXPECT_..."

Waarom ?

Veiligheidsmaatregel:
gecontroleerde groei

Hoe ?

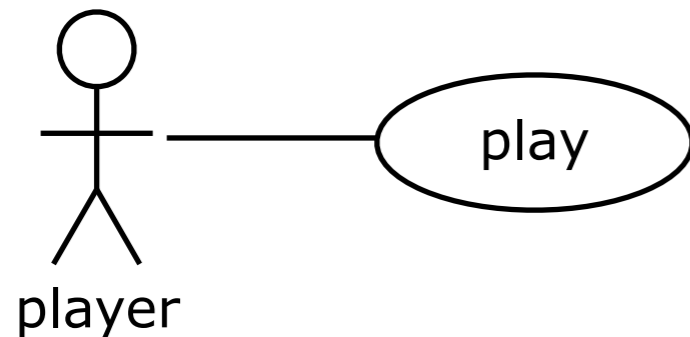
Pas bestaande tests aan

Tests groeien mee ...

```
// Tests whether a given string of moves is legal
TEST_F(TicTactToeTest, LegalMoves) {
    EXPECT_TRUE(tttPlayer_.properlyInitialized());
    EXPECT_TRUE(legalMoves(""));
    EXPECT_TRUE(legalMoves("a1"));
    ...
    EXPECT_TRUE(legalMoves("a1a2a3b1b2b3c1c2c3"));
    //illegal moves
    EXPECT_FALSE(legalMoves(" "));
    ...
    EXPECT_FALSE(legalMoves("b1c4a1"));
}

// Tests the "happy day" scenario for a player
TEST_F(TicTactToeTest, HappyDayPlayer) {
    EXPECT_TRUE(tttPlayer_.properlyInitialized());
    tttPlayer_.setMarker('O');
    EXPECT_EQ('O', tttPlayer_.getMarker());
    ...
}
```

TicTacToe17



Announce winner
Waar voegen we dit bij ?

Use Case 1: play

- Goal: 2 players play TicTacToe, 1 should win
- Precondition: An empty 3x3 board
- Success end: 1 player is the winner

Steps

1. Two players start up a game (First is "O"; other is "X")
2. WHILE game not done
 - 2.1 Current player makes move
 - 2.2 Switch current player
3. Announce winner

Conclusie

Eerst moet je het doen

- KISS principle: klein beginnen en langzaam groeien
- tests lopen + contracten worden nageleefd

Daarna moet je het *goed* doen

- lage koppeling, hoge cohesie
- model van het probleemdomein

- en ... tests blijven lopen
+ contracten blijven nageleefd

} goed ontwerp

Vuistregels

Ontwerpen

- “0, 1, infinity” principle
- Say things once and only once
- Plaats het gedrag dicht bij de data
- Nooit user-interface code in de basisklassen
 - *Zeker* niet voor invoer — frequent testen
- Maak een model van het probleemdomein (= het domeinmodel)
 - zelfstandige naamwoorden & werkwoorden als indicators



Testen

- Test lange uitvoer door het vergelijken van files
 - Test the hulpfuncties in de tests !
- Een test verwacht géén gebruikersinvoer

Evaluatie Criteria (Tests)

Inleiding Software Engineering (20?? - 20??)

Tweede Evaluatie

Student1: student1 - Student2: student2 - Student3: student3

Commentaar:

Tests

Nt aanwezig
(= geen tests)

Vrkrd. gebruik
(= manuele test)

Beperkt
(= nt. alles getest)

Voldoende
(= geen uitvoer, verkeerd invoer)

Goed
(= normale test)

Excellent
(= testcode goed)

Een test verwacht géén gebruikersinvoer

Test lange uitvoer door het vergelijken van files

Evaluatie Criteria (Ontwerp)

Objectgericht Ontwerp (Alles aankruisen!)

- Geen
- Inheritance
- Reuse Lijsten
- Goeie ADT
- Obj. collaboratie
- Controle Obj. printobjecten/iterators

Data encapsulatie

- geen
- aparte files
- rechtstreekse access
- getters/setters

Smelly code

- Long method
 - Long parameter list
 - Inappropriate intimacy
- bv.:
bv.:
bv.:

Gefundeerde beslissingen

- Geen reden
- intuïtie
- doordacht

Maak een model van het probleemdomain

Nooit user-interface code in de basisklassen

Plaats het gedrag dicht bij de data

"0, 1, infinity" principle
Say things once and only once