13주차 예비보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20191599 이름: 송경호

**1.**

Finite State Machine, 줄여서 FSM은 말 그대로 유한한 상태를 가질 수 있는 기계를 말한다. FSM은 I / O / S 세 가지 집합으로 구성되는데 각 각 입력에 대한 집합, 출력에 대한 집합, 현재 상태에 대한 집합을 의미한다. 각 상태는 외부 사건에 따라 계속해서 변화하며 이를 Transition 전이라고 한다.

FSM은 Sequential Circuit의 설계에 이용되는 데 그 과정은 아래와 같다.

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

먼저, problem description 단계로 문제를 정의하며 각 상태를 이진수의 형태로 정의하는 State Encoding 과정을 거친다. 예를 들어, 4개의 상태가 있는 FSM의 경우 각 state는 00, 01, 10, 11로 나타낼 수 있다. 때문에 N개의 상태 개수가 있을 때, 이를 log2N bit로 나타낼 수 있다. 이때, FSM의 최적화를 위해선 State Encoding을 최적화하여 사용 비트 수를 줄이는 것이 중요하다. 이를 통해 회로의 복잡성 및 크기를 줄일 수 있다.

다음으로, 각 state와 state 간의 상태 전이를 시각적으로 표현하는 State Graph를 그린다. 여기서 각 state는 노드로, 상태 전이는 화살표로 나타내는 것이 일반적이다. State Table은 상태 전이와 state를 표를 이용해서 나타낸 것이다. 표에는 현재 상태, 입력, 다음 상태, 출력 등을 기록하며 이는 결과적으로 state graph와 동일하게 전체의 과정을 시각적으로 표현하는 과정이지만, state table로 변환하는 것이 보다 정보를 명확하게 정리하는 것에 도움이 된다.

다음으로 state table을 기반으로, Flip-Flop의 입력을 결정한다. 일반적으로 state의 각 비트가 Flip-Flop의 상태로 사용되며, 상태 전이에 따라 해당 비트들이 업데이트 되는 형태로 Flip-Flop으로의 변환을 진행한다.

FSM은 디지털 회로 설계 뿐만 아닌, 제어 시스템 통신 프로토콜 등 다양한 응용 분야에서 사용된다. 또한 대표적인 FSM은 뒤에 다룰 Mealy machine과 Moore machine이 있다.

**2.**

Mealy Machine은 현재 상태와 입력을 바탕으로 출력이 결정되는 FSM을 의미한다.

텍스트, 도표, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위와 같이, Comb Logic회로의 입력으로써 present state와 present input, 즉 현 상태와 입력을 받아 출력이 발생하는 것을 알 수 있다. Mealy Machine의 예시는 아래와 같다.

3개의 연속된 1이 입력으로 들어오면 1을 출력하는 Mealy Machine은 아래와 같은 State Graph를 따른다.

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

앞서 말했듯이 State Graph에서 State는 노드를 상태 전이는 화살표를 이용해 나타낸다. 따라서 해당 FSM은 3개의 state A, B, C를 갖는다. 이때 A는 이전의 입력이 0인 상태, B는 이전의 입력이 1인 상태, 마지막으로 C는 이전의 두 입력이 모두 1인 상태를 의미한다. 또한 a/b와 같은 숫자에서 a는 입력 b는 출력을 의미하게 된다. Mealy Machine의 상태 변화는 현 상태와 입력에 따라 결정되므로 먼저 현 상태가 A인 경우 0입력시 0을 출력하고 state의 변화는 발생하지 않게 된다. A의 정의가 이전의 입력이 0인 상태이므로 이는 매우 직관적인 결과이다. 때문에 만약에 1이 입력될 경우 출력은 여전히 0이지만 state는 B로 변화하게 된다. 다음으로 B에서 0이 입력될 경우 출력 0과 함께 다시 A 상태로 변화하게 되며 1이 입력될 경우 아직 연속된 두번의 1만이 입력된 것이므로 출력은 0이지만 상태는 C로 변화한다. 마지막으로 C의 경우, B와 마찬가지로 0이 입력될 경우 출력 0과 함께 다시 A 상태로 변화하게 되지만 1의 입력의 경우 충족된 조건을 모두 만족했으므로 1을 출력하게 된다. 또한 여전히 C의 정의인 이전의 두 입력이 모두 1인 상태 역시 충족하고 있으므로 상태의 변화는 발생하지 않는다.

위의 State Graph를 State Table로 변환한 결과이다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Current State** | **Next State** | | **Output** | |
| **Input 0** | **Input 1** | **Input 0** | **Input 1** |
| A | A | B | 0 | 0 |
| B | A | C | 0 | 0 |
| C | A | C | 0 | 1 |

State Graph보다 State Table이 훨씬 직관적임을 알 수 있다. 이 때, State의 개수는 총 3개이므로 이를 2 bit를 통해 나타낼 수 있으며, A, B, C에 차례대로 비트 00, 01, 10을 부여했을 때 D Flip-Flop을 이용한 State Table은 아래와 같아진다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **X** | **A\*** | **B\*** | **z** | **DA** | **DB** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |

A와 B는 위에서 설명한 State이며, 입력은 X, 출력은 Z이다.

이를 바탕으로 회로 구성을 위한 Boolean Function을 구해야 한다. 이제껏 그래왔듯이 카르노 맵을 이용한다. **z, DA, D­b**에 대한 카르노맵 은 아래와 같다.

1) z = AX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X / AB | 00 | 01 | 11 | 10 |
| 0 |  |  | X |  |
| 1 |  |  | X | 1 |

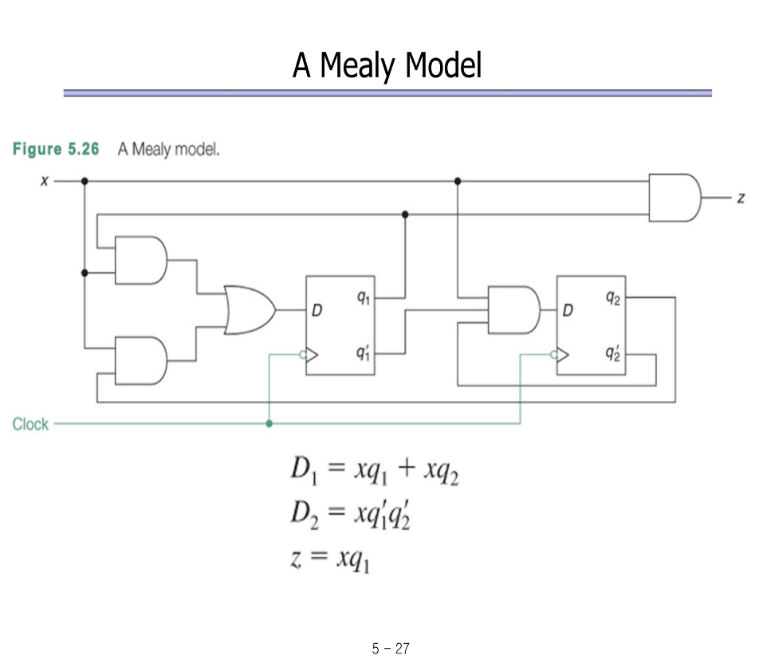
2) DA = AX + BX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X / AB | 00 | 01 | 11 | 10 |
| 0 |  |  | X |  |
| 1 |  | 1 | X | 1 |

3) DB = A’B’X

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X / AB | 00 | 01 | 11 | 10 |
| 0 |  |  | X |  |
| 1 | 1 |  | X |  |

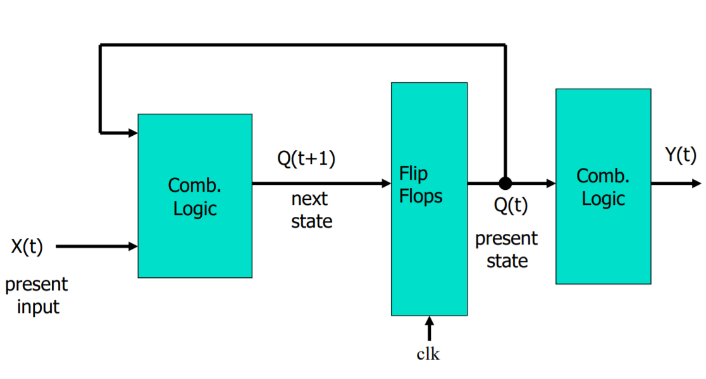
이를 바탕으로 회로 구성 시, 아래와 같은 형태를 띄게 된다.

****

앞서 설명했듯이, Mealy Machine은 현 state와 입력에 대해 출력이 결정된다. 회로에서 볼 수 있듯이 출력 Z의 입력으로 A (현재 state)와 X (input)이 들어간다.

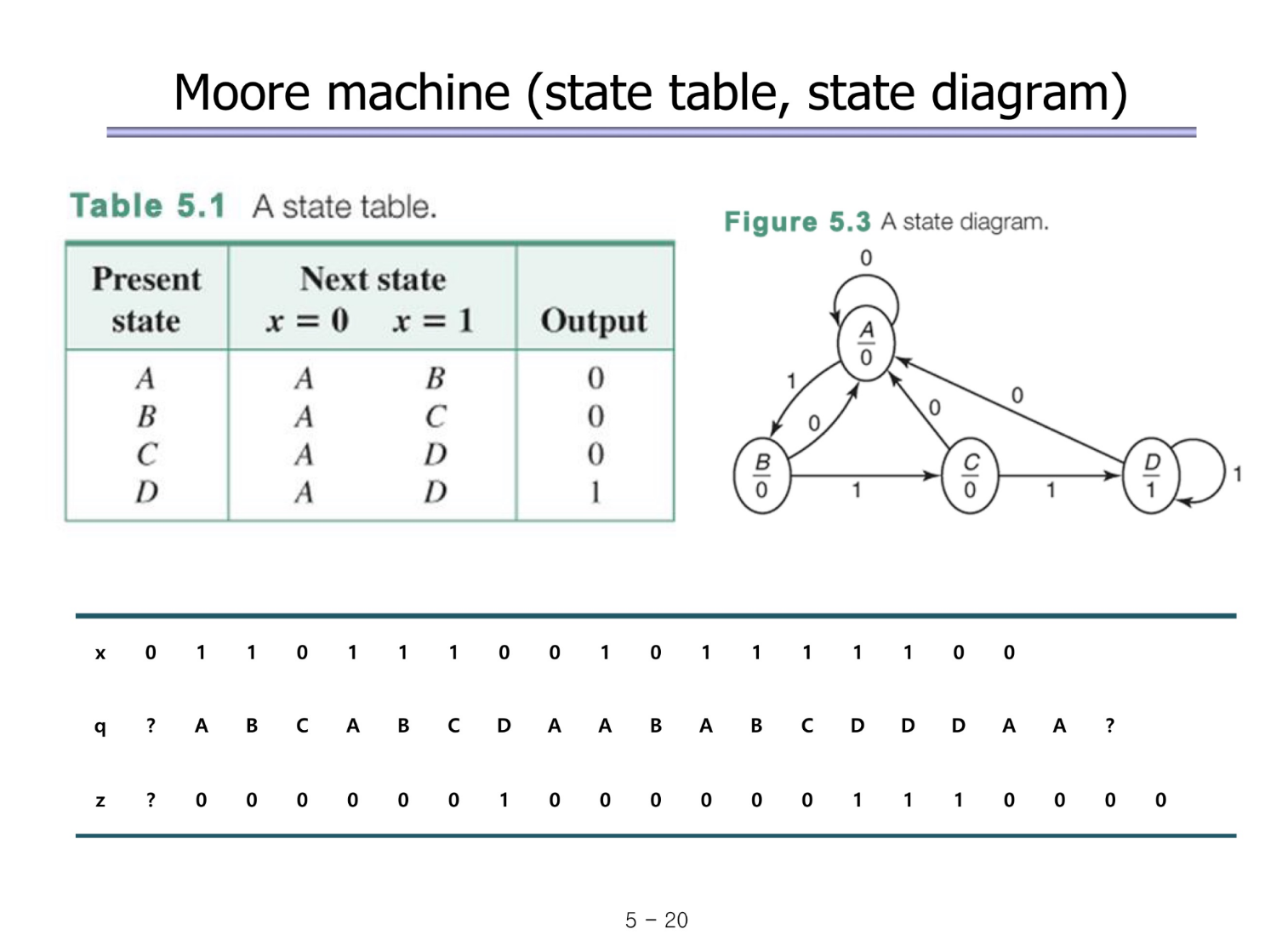
**3.**

Moore Machine은 Meally Machine과 달리 현재 상태만을 바탕으로 출력이 결정되는 FSM을 의미한다.

****

이전 Meally Machine과 달리 Comb Logic회로의 입력으로써 present state, 즉 현 상태만을 받아 출력이 발생하는 것을 알 수 있다. Moore Machine의 예시는 Meally Machine과 같은 예제를 이용한다.

3개의 연속된 1이 입력으로 들어오면 1을 출력하는 Moore Machine은 아래와 같은 State Graph를 따른다.



동일하게 노드는 State를 상태 전이는 화살표를 이용해서 나타냈다. 이때, Meally Machine의 State Graph와 달리 상태 전이 시, 입력에 따라 상태가 정의되지만 입력에 대해 출력이 발생하는 것이 아닌, state와 출력이 묶여있어 특정 state에서 출력이 정의되어 있다. State는 A, B, C, D 4개로 각 각 입력이 0인 상태, 입력이 1인 상태, 현 입력과 전 입력이 1인 상태, 현 입력과 두 개의 전 입력이 모두 1인 상태이다. 모든 State에서 0을 입력 받을 경우 State A로 상태 전이가 발생하며 이때 출력은 반드시 0이다. 또한 A에서 연속적으로 1을 입력 받을 때마다 State가 B > C > D순으로 변하는데 이때 B, C의 경우 모두 0의 출력을 갖지만 D는 조건이 충족되었으므로 1의 출력을 갖게 된다. 마지막으로 D에서 1을 입력 받을 경우 다시 D로 State가 변화하며 계속해서 1을 출력한다.

위의 State Graph를 State Table로 변환한 결과이다.

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Next State** | | **Output** |
| **Input X = 0** | **Input X = 1** |
| A | A | B | 0 |
| B | A | C | 0 |
| C | A | D | 0 |
| D | A | D | 1 |

Moore Machine의 경우 Meally Machine과 달리 State에 대해서만 Output이 발생하기 때문에 State Table이 조금 더 간단 해졌지만, 대신 State D가 하나 더 발생한 것을 알 수 있다. 같은 방법으로 State를 D Flip-Flop을 사용할 때의 Table로 변환하며 여전히 4개의 State이기 때문에 2 bit를 이용한다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **X** | **A\*** | **B\*** | **z** | **DA** | **DB** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

동일하게 Boolean Function을 구하기 위해 카르노맵을 구성한다.

1) z = AB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X / AB | 00 | 01 | 11 | 10 |
| 0 |  |  | 1 |  |
| 1 |  |  | 1 |  |

2) DA = (A+B)X

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X / AB | 00 | 01 | 11 | 10 |
| 0 |  |  |  |  |
| 1 |  | 1 | 1 | 1 |

3) DB = (A + B’)X

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X / AB | 00 | 01 | 11 | 10 |
| 0 |  |  |  |  |
| 1 | 1 |  | 1 | 1 |

이에 따라 아래와 같이 회로를 구성한다.

도표, 그림, 스케치, 라인이(가) 표시된 사진

자동 생성된 설명

Meally Machine과 달리 출력 z에 대하여 현 상태인 A와 B만이 사용되고 X는 입력되지 않는다.

**4.**

Sequence Detector란 문자열을 입력받아, 특정 타겟 sequence가 보일 때마다 출력 1을 생성하는 machine이다. 입력 문자열은 비트 문자열로 0과 1로만 이루어져 있으며, Sequence Detector는 overlap과 non-overlap 유형이 있다.

Overlap 유형은 중첩의 문자열 등장을 허용하는 Sequence Detector이다. 따라서 이전의 Sequence의 비트가 다른 Sequence의 비트로 포함될 수 있다. 예를 들어 101 Sequence를 검출하는 Overlap Sequence Detector의 경우, 10101이라는 문자열에 대해 101이 2번 나타난다고 판단한다.

반대로 NonOverlap 유형은 문자열의 중첩을 허용하지 않는 Sequence Detector이다. 따라서 이전과 달리 특정 Sequence의 비트가 다른 Sequence의 비트에 포함될 수 없다. 따라서 101 Sequence를 검출하는 NonOverlap Sequence Detector은 위의 10101의 문자열에 대해 101이 1번 나타난다고 판단한다. 만약 확인하는 전체 문자열이 1010101의 경우에 대해서 101이 2번 나타난다고 판단할 수 있다.

앞서 설명한 연속된 3번의 1을 확인하는 Meally와 Moore Machine은 111 Sequence Detector에 대한 FSM이다. 이때, 이 경우 overlap 유형이 된다.

1) Mealy Machine으로 검출

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X** | **0** | **1** | **1** | **0** | **1** | **1** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **0** | **0** |  |  |
| **q** | - | A | B | C | A | B | C | C | A | A | B | A | B | A | B | C | C | C | C | A | A |  |
| **Z** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

2) Moore Machine으로 검출

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X** | **0** | **1** | **1** | **0** | **1** | **1** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **0** | **0** |  |  |  |
| **q** | - | A | B | C | A | B | C | D | A | A | B | A | B | A | B | C | D | D | D | A | A | - |  |
| **Z** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

두 Machine의 작동 방식은 다르지만 출력 Z를 봤을 때 1의 개수가 동일한 것을 알 수 있다. 두 machine 모두 출력이 1일 경우 연속된 3개의 1이 발생한 것이므로 동일하게 작동한 것을 알 수 있다.

**5.**

**5-1) Mealy VS Moore**

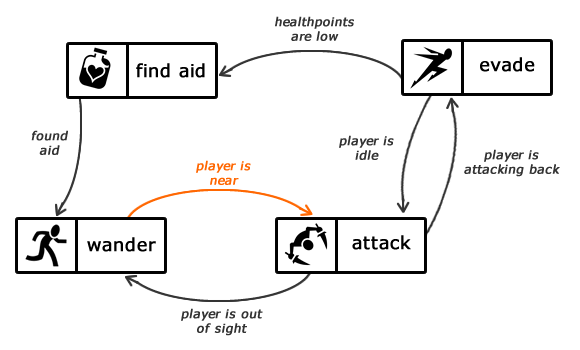
Moore Machine 방식의 설계는 Mealy보다 직관적인 경우가 있다. 현 상태에 의해서만 결과값이 출력되므로 입력과 출력이 직관적인 형태로 발생한다. 반대로 Mealy FSM은 Moore에 비해 덜 직관적이고 복잡한 경우가 있다.

그러나 이전 예시를 보면 알 수 있듯이, Mealy 방식 사용 시, 보다 적은 State를 이용해 설계가 가능하다. State가 적은 것은 회로의 설계에 큰 영향을 미칠 수 있으므로, Moore FSM을 생성하고 이를 기준으로 Mealy FSM으로 최적화하는 과정을 갖는 것이 효과적으로 FSM을 설계하는 방식이 될 수 있다.

**5-2) FSM의 활용**

FSM은 상태 변화를 효과적으로 나타낼 수 있어, 실생활 속 다양한 소프트웨어에서 사용된다. 게임의 에이전트의 경우 FSM을 활용하여 설계 가능하다.

게임 속에서, 다양한 입력에 대해 다양한 상태를 고려해야 되는 경우가 많다. FSM은 게임 내에 모든 시간에 처할 수 있는 모든 상태를 갖고 있으며, 입력에 따라 액션(출력)을 반환하고, 상태를 전환하게 된다.



위의 그림과 같이 게임의 흐름을 FSM을 활용하여 나타낼 수 있다. 이를테면, 해당 FSM은 게임 내의 몬스터와 같은 객체의 FSM이다. Player를 발견할 때 까지 돌아다니가, player를 발견하면 공격하고, player가 반격하면 도망치고 회복한다.