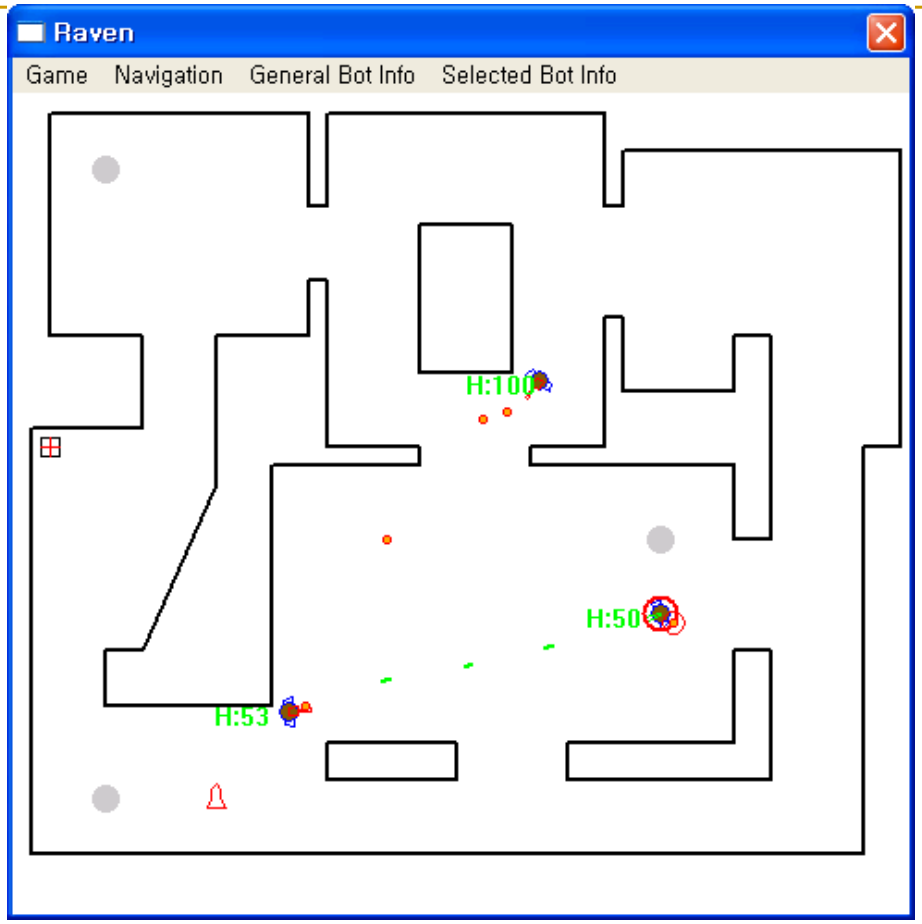
Chapter 7

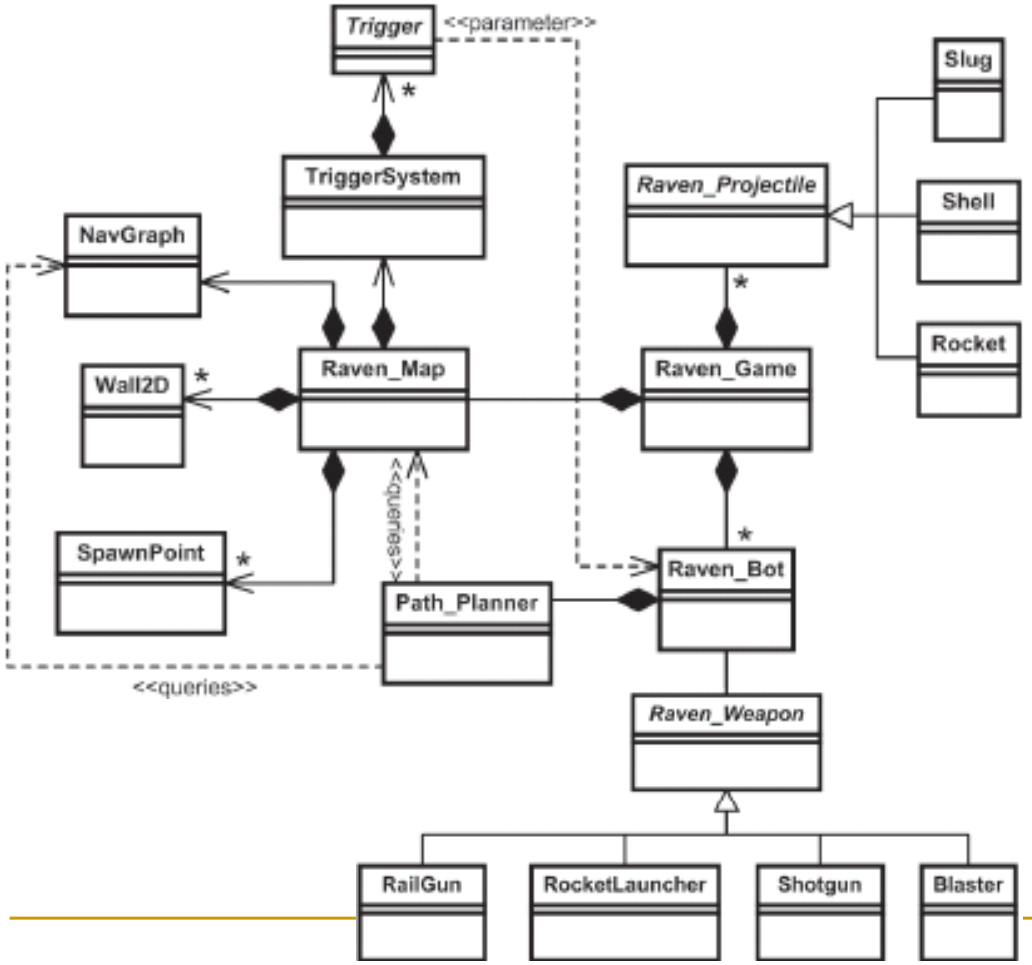
Raven



- lua.org
  - 최신 버전의 library를 download
    - https://sourceforge.net/projects/luabinaries/files/
      - **Windows Libraries \ Static \ lua-5.4.2_Win32_vc16_lib.zip**
    - Common 밑의 lua-5.1.3 대체
      luahelperfunctions.h, OpenLuaStates.h 기존 폴더에서 복사
    - '프로젝트 속성 \ VC++ 디렉토리'에서
      - '포함 디렉토리', '라이브러리 디렉토리' 변경
    - scriptor.h

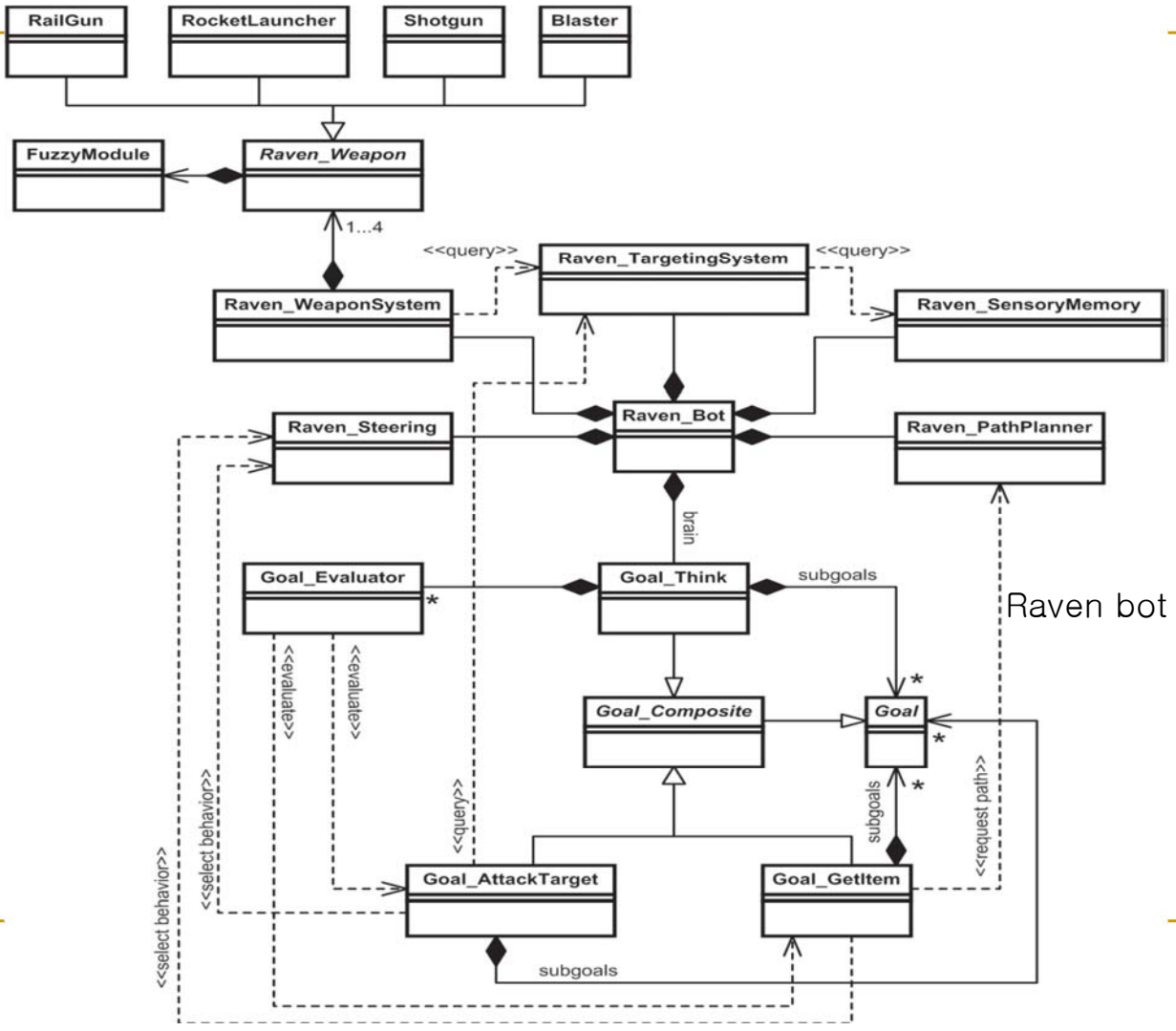
```
#pragma comment(lib, "lua54.lib")
//#pragma comment(lib, "lua5.1.lib")
```


- UI
  - bot은 우클릭으로 선택, 빨간원
    - 선택된 bot에 우클릭 -> 파란원, 소유, 사용자 play
    - 해제: 다른 bot을 우클릭하거나 'X' 키 입력
  - 이동: 맵에 우클릭
  - 발사: 좌클릭
    - '1' ~ '4' : 무기 선택



RailGun Ammo

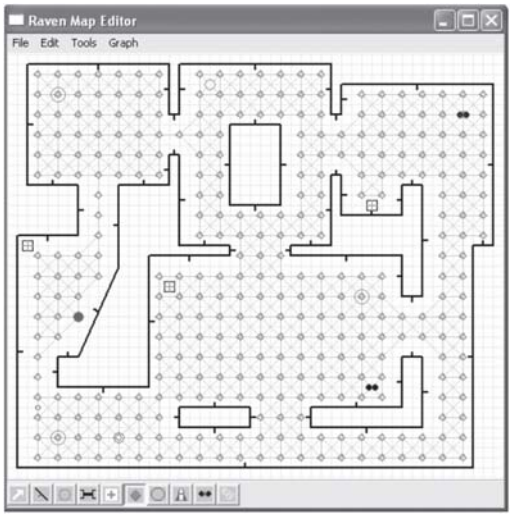
Raven 아키텍처



Raven bot AI

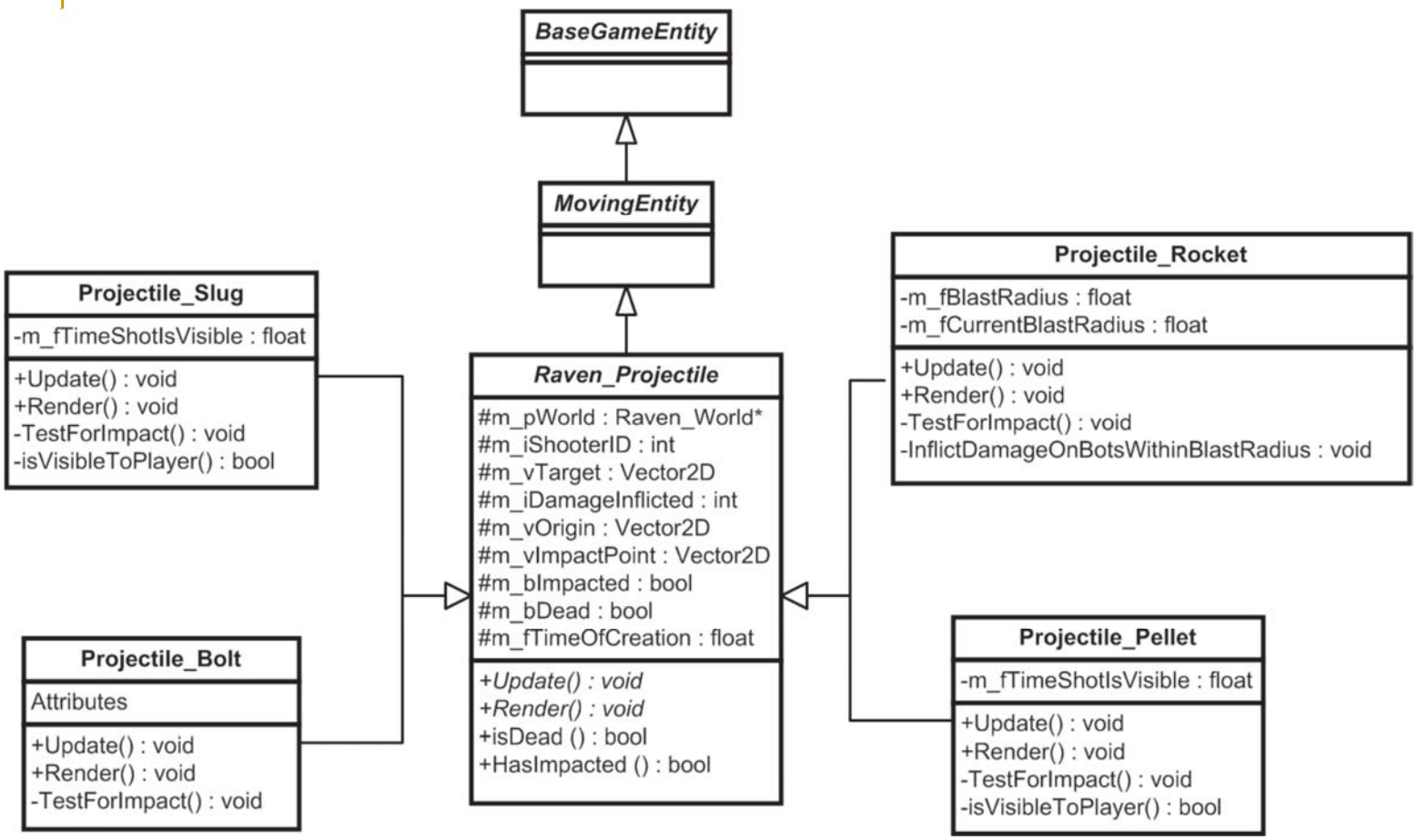
class Raven_Game

```
class Raven_Game { // 프로젝트의 허브
private:
  Raven_Map*                 m_pMap;
  std::list<Raven_Bot*>      m_Bots;
  Raven_Bot*                 m_pSelectedBot;
  std::list<Raven_Projectile*>  m_Projectiles;
public:
  void Render();
  void Update();
  bool LoadMap(const std::string& FileName);
  bool isPathObstructed(Vector2D A, Vector2D B, double BoundingRadius = 0)const;
  std::vector<Raven_Bot*> GetAllBotsInFOV(const Raven_Bot* pBot)const;
  bool       isSecondVisibleToFirst(const Raven_Bot* pFirst,
                                    const Raven_Bot* pSecond)const;
};
```

Raven 맵 에디터

```
class Raven_Weapon
{
 Raven_Weapon(unsigned int TypeOfGun,
              unsigned int DefaultNumRounds,
              unsigned int MaxRoundsCarried,
              double       RateOfFire,
              double       IdealRange,
              double       ProjectileSpeed,
              Raven_Bot*   OwnerOfGun);
 bool          AimAt(Vector2D target)const;
 virtual void  ShootAt(Vector2D pos) = 0;
 virtual void  Render() = 0;
 virtual double GetDesirability(double DistToTarget)=0;
 double         GetMaxProjectileSpeed();
 int           NumRoundsRemaining();
 void          DecrementNumRounds();
 void          IncrementRounds(int num);
 unsigned int  GetType();
};
```

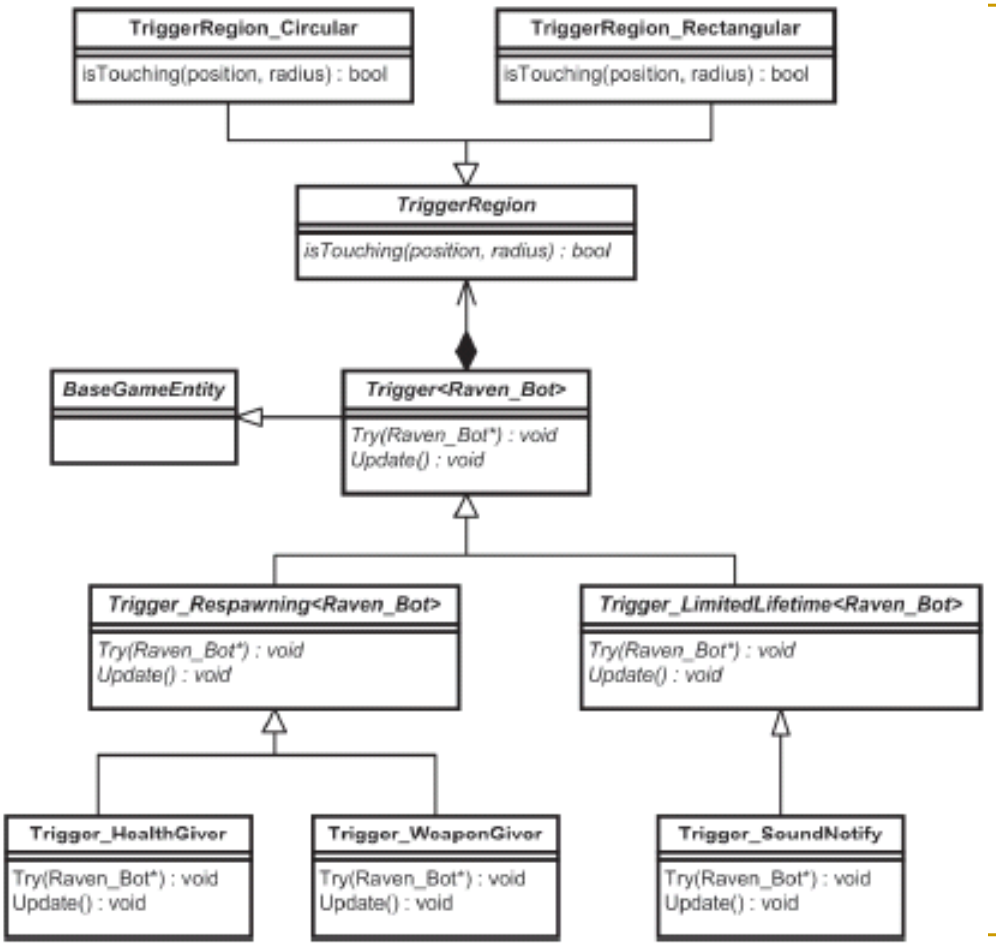



class Raven_Map

```
class Raven_Map {
public:
  typedef NavGraphNode<Trigger<Raven_Bot>*>        GraphNode;
  typedef SparseGraph<GraphNode, NavGraphEdge>     NavGraph;
  typedef Trigger<Raven_Bot>                       TriggerType;
  typedef TriggerSystem<TriggerType>               TriggerSystem;
private:
  std::vector<Wall2D*>              m_Walls;
  TriggerSystem                     m_TriggerSystem;
  std::vector<Vector2D>             m_SpawnPoints;
  //this map's accompanying navigation graph
  NavGraph*                         m_pNavGraph;
public:
void    AddSoundTrigger(Raven_Bot* pSoundSource, double range);
  double   CalculateCostToTravelBetweenNodes(int nd1, int nd2)const;
  void  UpdateTriggerSystem(std::list<Raven_Bot*>& bots);
};
```


트리거

- 조건을 정의하는 객체



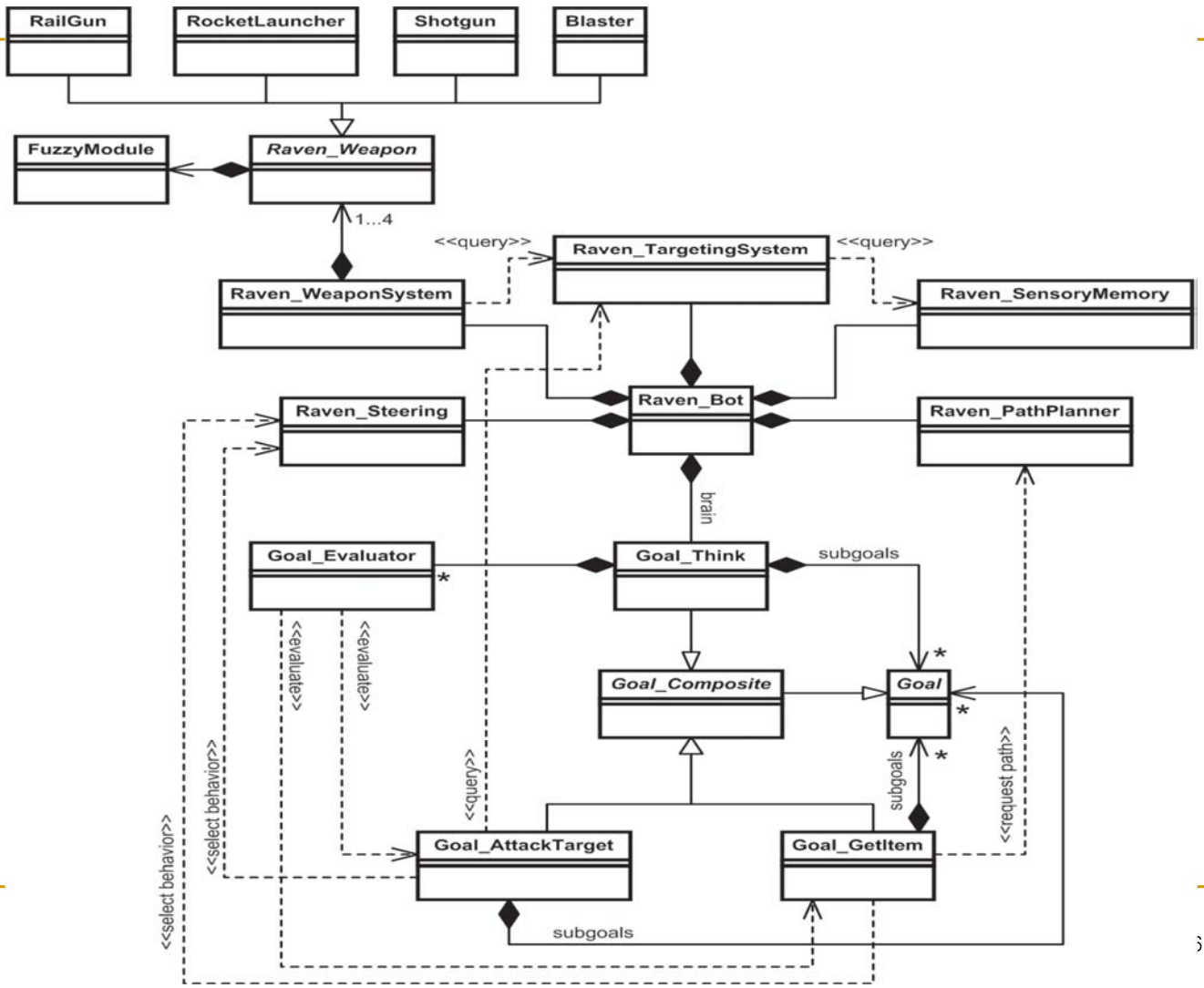
```
template <class trigger_type>
class TriggerSystem {
typedef std::list<trigger_type*> TriggerList;
TriggerList   m_Triggers;
 void UpdateTriggers()  {
    TriggerList::iterator curTrg = m_Triggers.begin();
    while (curTrg != m_Triggers.end())   {
      //remove trigger if dead
      if ((*curTrg)->isToBeRemoved())    {
        delete *curTrg;
        curTrg = m_Triggers.erase(curTrg);
      }
      else    {
        //update this trigger
        (*curTrg)->Update();
        ++curTrg;
      }
    }
  }
```



```
 template <class ContainerOfEntities>
  void TryTriggers(ContainerOfEntities& entities) {
    //test each entity against the triggers
    ContainerOfEntities::iterator curEnt = entities.begin();
    for (curEnt; curEnt != entities.end(); ++curEnt)  {
      //an entity must be ready for its next trigger update and it must be
      //alive before it is tested against each trigger.
      if ((*curEnt)->isReadyForTriggerUpdate() && (*curEnt)->isAlive()) {
        TriggerList::const_iterator curTrg;
        for (curTrg = m_Triggers.begin(); curTrg != m_Triggers.end();
++curTrg) {
          (*curTrg)->Try(*curEnt);
        }
      }
    }
  }
 void Update(ContainerOfEntities& entities)
  {
    UpdateTriggers();
    TryTriggers(entities);
  }
};
```

Raven bot AI

- 의사 결정 : 목표 중재 (arbitration of goal, Goal_Think)
- 이동: 조종 행동 (Raven_Bot <- MovingEntity)
- 길 계획하기: 목표 위치로 이동, 아이템 획득
- 지각
  - class MemoryRecord
  - class Raven_SensoryMemory
    typedef std::map<Raven_Bot*, MemoryRecord> MemoryMap;
- 목표 선택: Raven_TargetingSystem
- 무기 다루기: Raven_WeponSystem
    typedef std::map<int, Raven_Weapon*> WeaponMap;



AI 구성요소의 갱신

→ 엑세스 횟수의 갱신 주기를 바꿔주는 역할

- 각각의 갱신을 조절하기 위해 Regulator 사용

```
class Regulator {
  double m_dUpdatePeriod; //the time period between updates
  DWORD m_dwNextUpdateTime; //the next time the regulator allows code flow
  Regulator(double NumUpdatesPerSecondRqd)  {
    m_dwNextUpdateTime = (DWORD)(timeGetTime()+RandFloat()*1000);
    if (NumUpdatesPerSecondRqd > 0)    {
      m_dUpdatePeriod = 1000.0 / NumUpdatesPerSecondRqd;
    }
    else if (isEqual(0.0, NumUpdatesPerSecondRqd))    {
      m_dUpdatePeriod = 0.0;
    }
    else if (NumUpdatesPerSecondRqd < 0)    {
      m_dUpdatePeriod = -1;
    }
  }
  //returns true if the current time exceeds m_dwNextUpdateTime
  bool isReady();
};
```



```
void Raven_Bot::Update()
{
  //process the currently active goal. Note this is required even if the bot
  //is under user control. This is because a goal is created whenever a user
  //clicks on an area of the map that necessitates a path planning request.
  m_pBrain->Process();
  //Calculate the steering force and update the bot's velocity and position
  UpdateMovement();
  //if the bot is under AI control but not scripted
  if (!isPossessed())  {
    //examine all the opponents in the bots sensory memory and select one
    //to be the current target
    if (m_pTargetSelectionRegulator->isReady())
    {
      m_pTargSys->Update();
    }
    //appraise and arbitrate between all possible high level goals
    if (m_pGoalArbitrationRegulator->isReady())
    {
       m_pBrain->Arbitrate();
    }
```

```
    //update the sensory memory with any visual stimulus
    if (m_pVisionUpdateRegulator->isReady())
    {
      m_pSensoryMem->UpdateVision();
    }

    //select the appropriate weapon to use from the weapons currently in
    //the inventory
    if (m_pWeaponSelectionRegulator->isReady())
    {
      m_pWeaponSys->SelectWeapon();
    }

    //this method aims the bot's current weapon at the current target
    //and takes a shot if a shot is possible
    m_pWeaponSys->TakeAimAndShoot();
  }
}
```


실습

- 전투
  - 하나의 bot을 선택하여 두 개 이상의 bot을 kill
  - www.lua.org
- Raven_Game::Update() 분석
- Memory record에 피해 정도 구현
  - bot이 피격되었을 때 감지할 수 있도록 감각시스템을 갱신시키는 코드를 작성
  - MemoryRecord에 필드를 만들어서 각 적이 입은 피해 정도를 기록하고 이를 목표 선택 기준의 일부로 사용하라
- Bot의 목표선택 기준 다양화
  - 다른 목표선택 기준을 적용하라
  - 각각의 bot이 유일한 기준을 사용하여 서로 게임하게 만든 후 어느 bot이 최상으로 수행하는 지를 판별하라
  - 적이 bot의 지향 방향에 대해 틀어진 각도, 적의 대면 방향, 적 무기 사거리, bot 무기 사거리, 생명치, 적이 얼마 동안 보였는가, 적이 bot에게 입힌 피해정도, 적이 bot에게 죽은 횟수, bot이 적에게 죽은 횟수