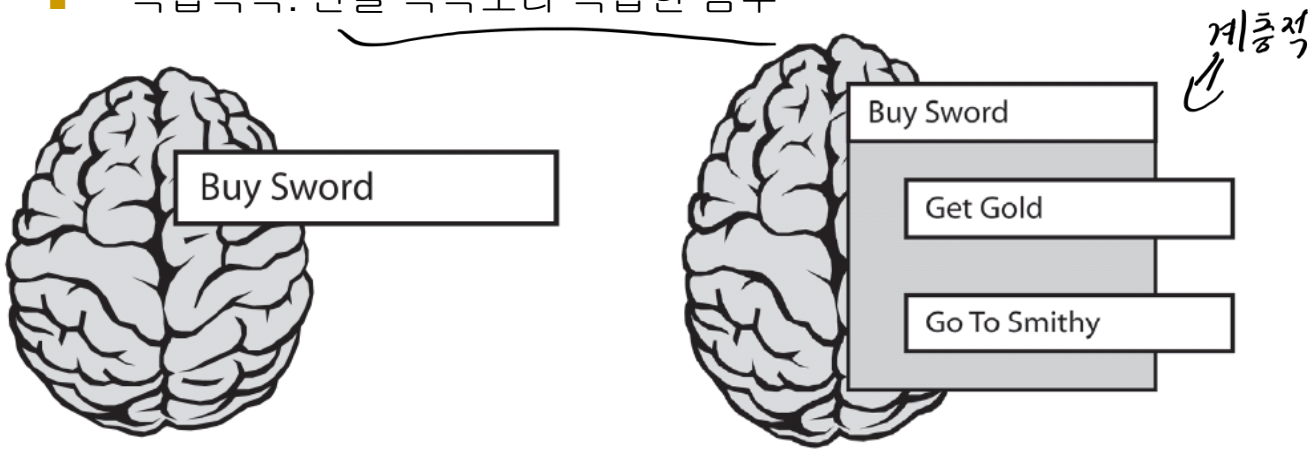
Chapter 9

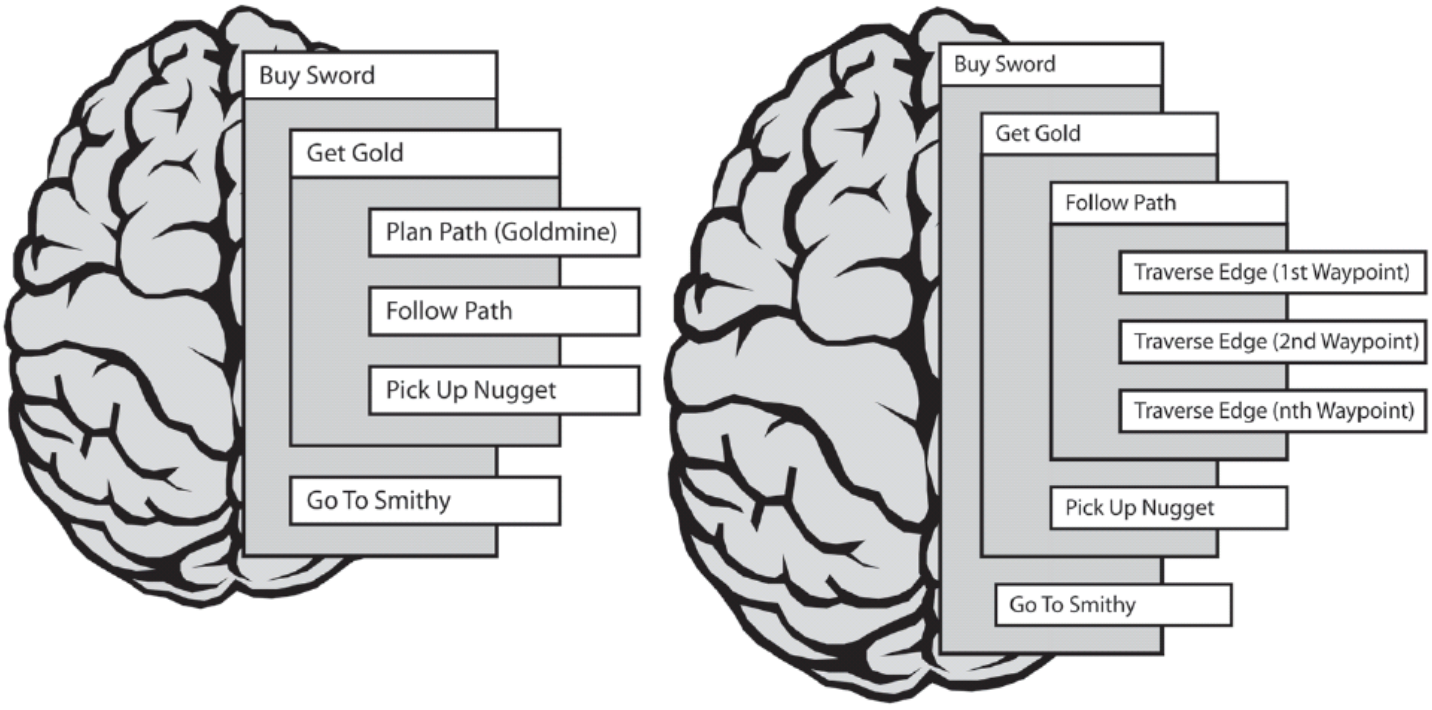
목적이 부여된 에이전트의 행동

계층적 목적(goal)

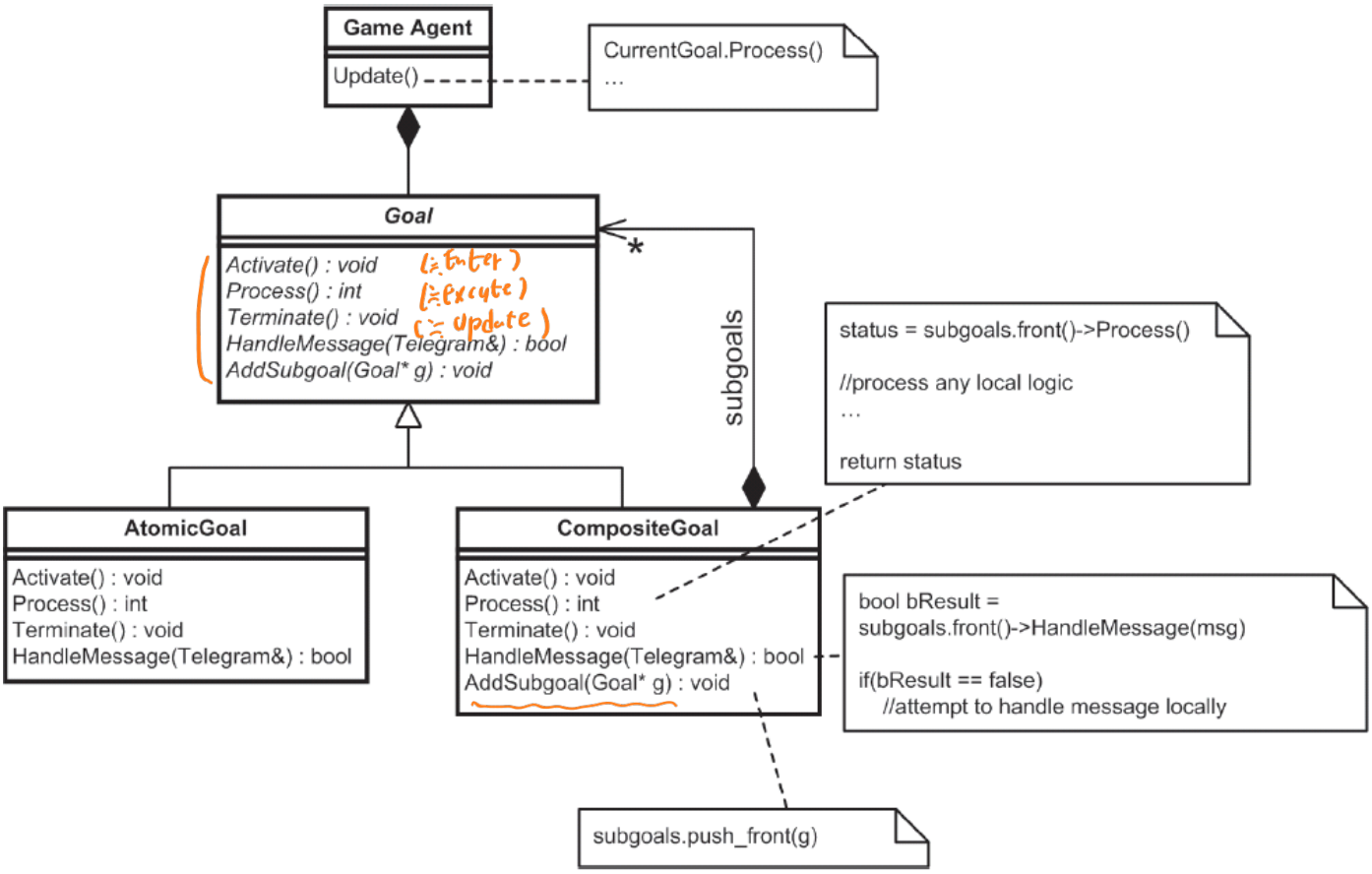
- 에이전트의 행동
  - 상태 대신에 계층적인 목적(goal)의 모음으로 정의
    - 단일목적: seek나 무기 재장전과 같은 단일 임무 및 행위를 정의
    - 복합목적: 단일 목적보다 복합한 임무



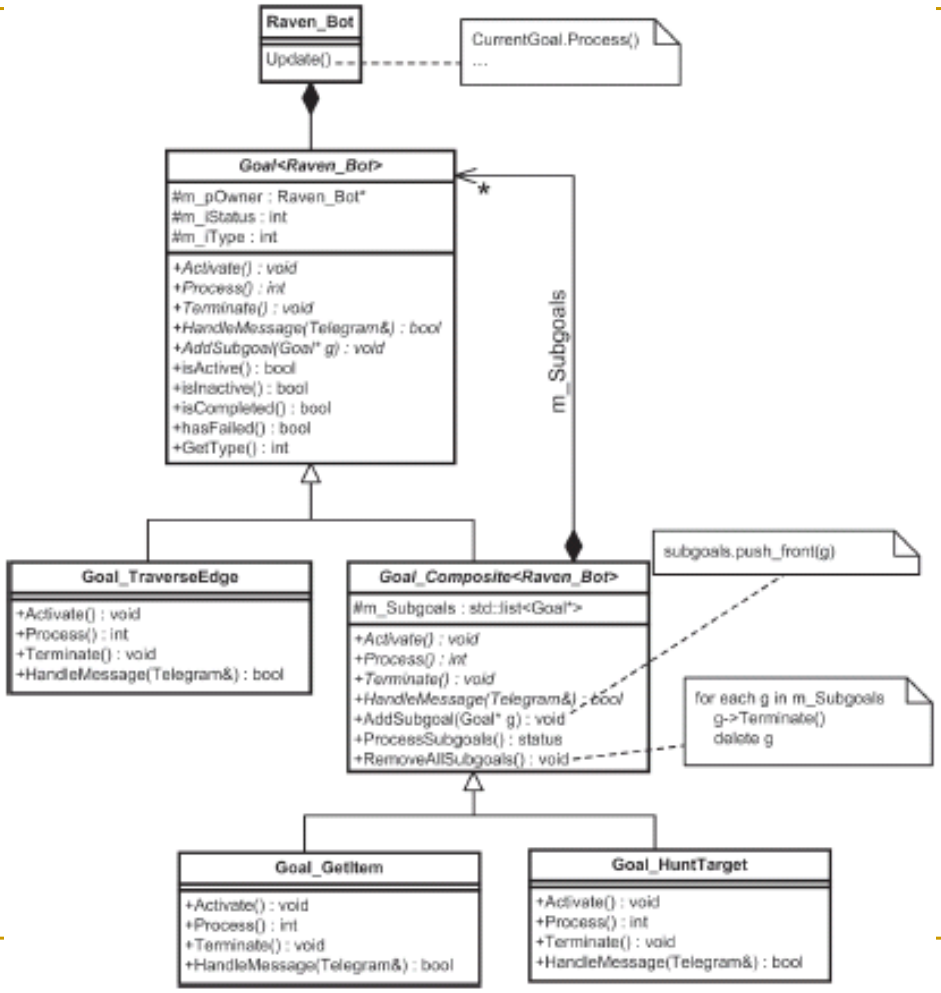
- 목적이 부여된 에이전트의 행동
  - goal_think가 갱신될 때마다 에이전트는 게임의 상태를 시험하고 일련의 미리 정의된 상위 레벨 목적이나 전략에서 하나를 선택
  - 그 다음에 이 목적을 쫓아 모든 구성하는 부목적으로 나누고, 각각을 차례대로 만족시켜 나감
  - 이 과정은 목적이 만족될 때까지, 실패할 때까지 아니면 게임 상태가 전략의 수정을 필요로 할 때까지 계속됨



복합 디자인 패턴



- void Activate()
  - 초기화 논리를 가지고 goal의 계획 단계를 나타냄
  - goal은 상황에 따라 다시 계획하기 위해 여러 번 Activate() 메소드를 호출
- int Process()
  - 매 update 단계마다 실행됨
  - goal의 상태를 나타내는 열거형 값을 반환
    - inactive, active, completed, failed
- void Terminate()
  - goal이 종료되기 전에 필요한 정리 작업을 수행



- Goal_Composite::ProcessSubgoals
  - 모든 복합 목적이 매 갱신 단계마다 이 메소드 호출
  - 완료되거나 실패한 모든 goal을 제거하고
    - 줄 서있는 다음 부목적을 처리
  - 부목적 리스트가 비어있다면 completed를 반환
- Goal_Composite::RemoveAllSubgoals
  - 부목적 리스트를 지움

```
template <class entity_type>
int Goal_Composite<entity_type>::ProcessSubgoals()
{
  //remove all completed and failed goals from the front of the subgoal list
  while (!m_SubGoals.empty() &&
         (m_SubGoals.front()->isComplete() || m_SubGoals.front()->hasFailed()))
  {
    m_SubGoals.front()->Terminate();
    delete m_SubGoals.front();
    m_SubGoals.pop_front();
  }
```



```
  //if any subgoals remain, process the one at the front of the list
  if (!m_SubGoals.empty())
  {
    //grab the status of the front-most subgoal
    int StatusOfSubGoals = m_SubGoals.front()->Process();
    //we have to test for the special case where the front-most subgoal
    //reports 'completed' *and* the subgoal list contains additional goals.When
    //this is the case, to ensure the parent keeps processing its subgoal list
    //we must return the 'active' status.
    if (StatusOfSubGoals == completed && m_SubGoals.size() > 1)
    {
      return active;
    }
    return StatusOfSubGoals;
  }
  //no more subgoals to process - return 'completed'
  else
  {
    return completed;
  }
}
```

```
template <class entity_type>
void Goal_Composite<entity_type>::RemoveAllSubgoals()
{
  for (SubgoalList::iterator it = m_SubGoals.begin();
       it != m_SubGoals.end();
       ++it)
  {
    (*it)->Terminate();
    delete *it;
  }
  m_SubGoals.clear();
}

template <class entity_type>
void Goal_Composite<entity_type>::AddSubgoal(Goal<entity_type>* g)
{
  //add the new goal to the front of the list
  m_SubGoals.push_front(g);
}
```


Raven Bot에서 사용되는 goal

Composite Goals	Atomic Goals
Goal_Think	Goal_Wander
Goal_GetItem	Goal_SeekToPosition
Goal_MoveToPosition	Goal_TraverseEdge
Goal_FollowPath	Goal_DodgeSideToSide
Goal_AttackTarget	
Goal_Explore	
Goal_HuntTarget	

목적 중재

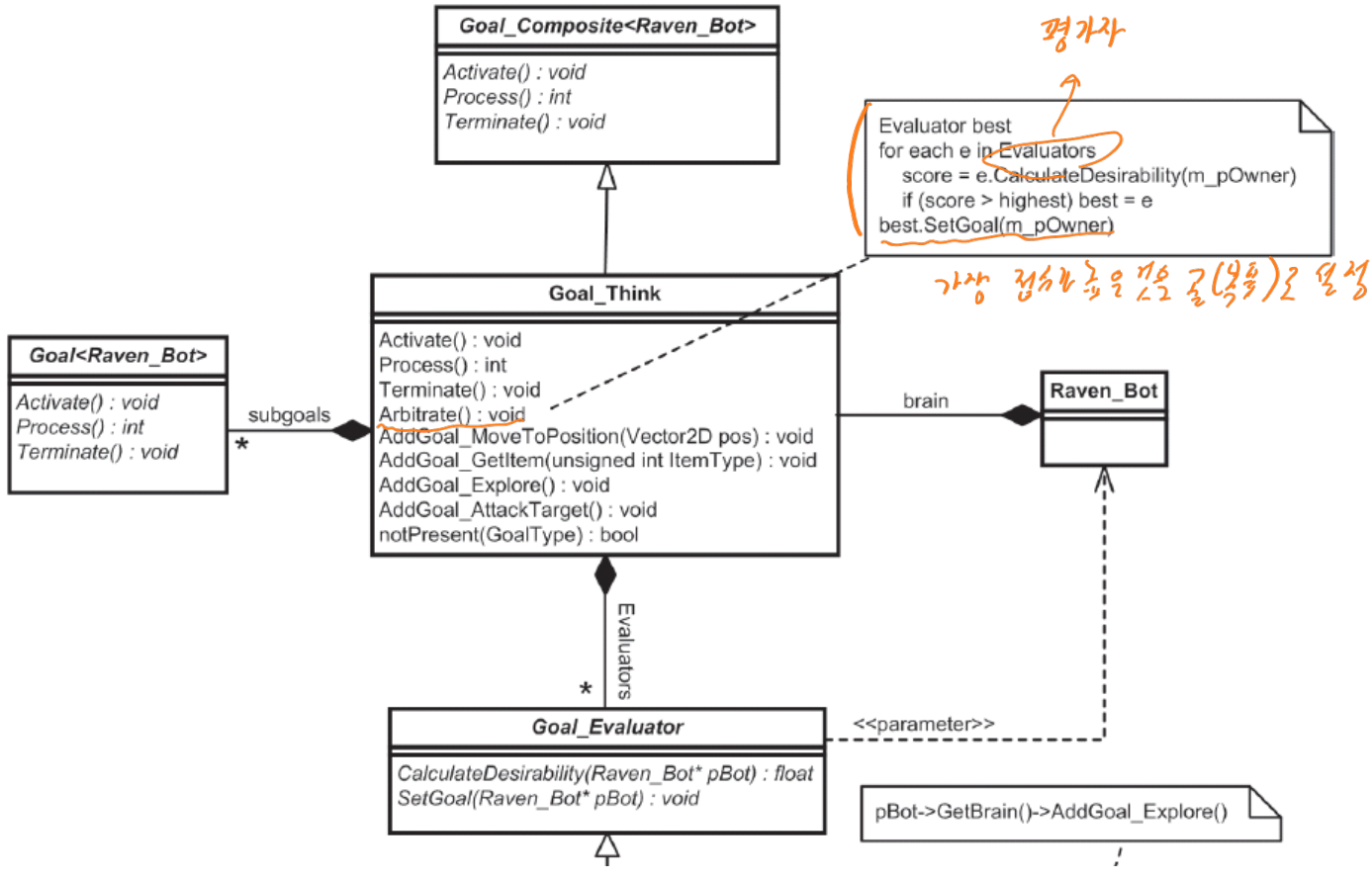
- Goal_Think
  - 최상위 레벨의 목적
  - 이용 가능한 전략들 중에서 가장 적합한 goal을 선택
  - 목적 중재는 본질적으로 몇 개의 숫자로 정의되는 알고리즘 과정
    - 결과적으로 논리(FSM과 같은)가 아닌 데이터로 운영됨
    - 행동을 변화시키기 위해서 단지 숫자들을 조정하면 됨

여섯 개의 전략 레벨 목적(goal)

- ExploreGoal_Evaluator
- GetHealthGoal_Evaluator
- GetWeaponGoal_Evaluator
  - Rocket
  - Shot Gun
  - Rail Gun
- AttackTargetGoal_Evaluator

class Raven_Feature

- 네 개의 특징 추출 함수 보유 [0, 1]
  - static double Health (Raven_Bot* pBot)
  - static double DistanceToItem (Raven_Bot* pBot, int ItemType)
  - static double IndividualWeaponStrength (Raven_Bot* pBot, int WeaponType)
  - static double TotalWeaponStrength (Raven_Bot* pBot)



```
void Goal_Think::Arbitrate()
{
  double best = 0;
  Goal_Evaluator* MostDesirable = 0;
  //iterate through all the evaluators to see which produces the highest score
  GoalEvaluators::iterator curDes = m_Evaluators.begin();
  for (curDes; curDes != m_Evaluators.end(); ++curDes)
  {
    double desirabilty = (*curDes)->CalculateDesirability(m_pOwner);
    if (desirabilty >= best)
    {
      best = desirabilty;
      MostDesirable = *curDes;
    }
  }
  MostDesirable->SetGoal(m_pOwner);
}
```



```
void AttackTargetGoal_Evaluator::SetGoal(Raven_Bot* pBot)
{
  pBot->GetBrain()->AddGoal_AttackTarget();
}
void Goal_Think::AddGoal_AttackTarget()
{
  if (notPresent(goal_attack_target))
  {
    RemoveAllSubgoals();
    AddSubgoal( new Goal_AttackTarget(m_pOwner));
  }
}
bool Goal_Think::notPresent(unsigned int GoalType)const
{
  if (!m_SubGoals.empty())
  {
    return m_SubGoals.front()->GetType() != GoalType;
  }
  return true;
}
```

과제

1. MemoryRecord에 피해 정도 등을 타겟팅시스템에 반영
   - 어떤 bot이 피격되었을 때 감지할 수 있도록 발사자의 감각 시스템을 갱신시키고 이 값을 발사자의 목표 선택 기준의 일부로 사용하라.
2. 부분 경로 생성(검색) 기능 추가
   - 사용자가 정의한 수의 검색 사이클이나 검색 깊이가 도달된 후에는 목표에 가장 가까운 노드에 이르는 경로를 반환하도록 A* 알고리즘을 변경
3. 전략 추가
   - 논리와 여분의 목적(goal), evaluator class, feature를 작성하라.
   - Goal_DodgeSideToSide 사용시 인정 안함

- 발표:
  - PPT로 알고리즘 설명
  - 프로그램 Demo.
- 제출: eclass.kpu.ac.kr
  - PPT: 알고리즘 설명과 실행 캡처 화면
  - 학번과 이름이 들어간 프로그램 소스