2019년도
학사학위논문

유전 알고리즘을 이용한 최적의 테트리스 AI 구현

Optimal Tetris AI implementation using Genetic Algorithm

2019년 11월 26일

순천향대학교 공과대학
컴퓨터공학과

장용민 김응민

2019년도
학사학위논문

유전 알고리즘을 이용한 최적의 테트리스 AI 구현

Optimal Tetris AI implementation using Genetic Algorithm

2019년 11월 26일

순천향대학교 공과대학
컴퓨터공학과

장용민 김응민

유전 알고리즘을 이용한
최적의 테트리스 AI 구현

Optimal Tetris AI
implementation using Genetic Algorithm

지도교수 이 해 각

이 논문을 공학사학위 논문으로 제출함

2019년 11월 26일

순천향대학교 공과대학
컴퓨터공학과

장용민 김응민

장 용 민

김 응 민 의 공학사학위논문을 인준함

2019년 11월 26일

심 사 위 원 이해각 인

심 사 위 원 하상호 인

심 사 위 원 남윤영 인

순천향대학교 공과대학

컴퓨터공학과

차 례

초 록

본 논문은 테트리스 게임 플레이를 대상으로 유전 알고리즘을 활용하여 최적화된 테트리스 게임 AI를 개발한다. 유전 알고리즘에 근거하여 필요한 가중치 요인을 결정하며, 그 중요도를 학습하고 이를 적용하여 테트리스 게임 플레이를 최적화하는 방법을 제시한다.

테트리스 게임 플레이는 사람마다 개인차가 있으나, 잘하는 플레이어의 경우 나름의 규칙이 있다. 이 규칙들을 추출하여 가중치 요소로써 변환하고, 검증하여 실제로 테트리스 AI에 학습시켰을 때, 플레이가 개선됨을 보이면 올바른 가중치 요소로 평가한다. 이렇게 올바르게 평가된 가중치들을 모아 하나의 적합도 함수를 구성하고, 실제 게임에서는 적합도 함수를 계산하여 블록의 위치를 결정한다.

하나의 유전자 풀은 여러 개의 유전자로 이루어져 있으며, 하나의 유전자는 여러 가중치를 갖는다. 이때 각 유전자의 가중치 계수들은 처음에 모두 무작위로 설정되어 있으며, 유전자 각각이 지닌 가중치 계수들과 실제 블록이 놓이는 위치를 곱하여 더한 것을 적합도로 두어 블록의 위치를 결정하여 놓는다. 하나의 유전자가 블록을 놓는 과정을 반복하여 게임이 끝날 경우, 유전자 풀의 다음 유전자가 게임을 플레이하며, 모든 유전자가 이 과정을 반복하면 하나의 세대가 끝난다.

각 세대가 끝날 때마다 게임 플레이에 더 최적화된 유전자들을 찾기 위해 유전자가 가진 가중치 계수들로 다음 세대를 위한 유전자들을 생성한다. 생성하는 방법에는 잘 알려진 선택 연산과 교차 연산, 돌연변이 연산을 사용한다.

위의 경과를 지켜보아 현재까지 진행된 가중치의 계수 값과 가중치 개수들에 따라 성능을 평가하고, 이를 통해 게임이 무한히 반복되는 수준의 AI를 구현한다.

주요어 : 유전 알고리즘, 최적화, 가중치, 적합도, AI

1. 서 론

1.1 연구 배경

최적화란 특정 집합 위에서 정의된 값이나 함수에 대해 그 값이 최대나 최소가 되는 상태를 해석하는 문제이다. 수학에서의 최적화는 어떤 제약조건이 있을 수도 있는 상황에서 최댓값이나 최솟값 또는 특정 값 같은 함수의 목푯값을 찾는 것이고, 컴퓨터과학에서의 최적화는 실행속도를 증가시키고 메모리 요구량을 감소시켜 시스템을 개선하는 것이다.

소프트웨어적으로 말하면 지정된 사양에 맞는 성능을 가지면서 신뢰성 있고, 유지 보수하기 쉬우며 변화하는 요구사항에 빠르게 적응할 수 있는 것을 말한다. 소프트웨어를 만들 때, 얼마만큼 최적화했는지에 따라 프로그램의 성능이 크게 차이가 나기 때문에 프로그래머에겐 최적화가 중요한 요소이다. 그렇기에 최적화와 관련된 기법에 관심이 있었다.

앞에서 설명한 최적화를 간단히 말하면 최적의 해 또는 방법을 찾는 것이다. 그러나 항상 모든 문제를 손쉽게 최적화할 수는 없다. 문제가 간단한 이차함수를 해석하는 것이나 행렬을 이용한 문제의 경우는 상대적으로 최적화하기에 편하지만, 복잡한 상황이 얽혀있는 문제의 경우 그러기가 쉽지 않다. 따라서 복잡한 상황에서의 최적화는 어떻게 할 수 있을까를 고민하다가 그 대상으로 주어진 조건(특정 조건을 만족하면 게임이 끝나는)이 있는 게임 플레이를 선정하였고 그것을 최적화하는 기법을 생각하였다.

최적화 알고리즘으로는 유전 알고리즘, 개미 군집 최적화 알고리즘, 담금질 기법, 화음 탐색법 등이 있으며, 본 논문에서 적용할 최적화 기법으로 유전 알고리즘을 사용하였다.

1.2 연구 목표

본 논문에서는 유전자 알고리즘을 게임에 적용해 최적의 플레이 방법을 찾는 것을 목표로 한다. 컴퓨터가 알고리즘에 따라 플레이 방법을 찾고, 그렇게 찾은 플레이 방법에 따라 게임을 할 때, 과연 그것이 최적의 플레이 방법인지 확인한다.

게임은 대중들에게 잘 알려진 블록 퍼즐게임인 테트리스로 선정하였다. 테트리스에 유전 알고리즘을 적용하여 최적의 플레이 방법을 찾고, 찾은 방법을 이용하여 게임을 하였을 시, 적어도 24시간 이상 게임이 끝나지 않고 계속 유지가 되었을 때 최적의 해를 찾았다고 여긴다.

1.3 연구 내용

알고리즘을 적용하기에 앞서 우선 테트리스 게임을 구현한 후에 플레이 방식을 연구한다. 테트리스는 2차원의 직사각형 틀 안에 서로 다른 모양의 블록들을 차곡차곡 쌓아서 최대한 오래 살아남는 것이 목표인 게임이다. 블록들을 차곡차곡 쌓아서 가로 한 줄 또는 한 줄 이상을 빈틈없이 채우게 되면 줄이 사라지며 점수가 오른다. 테트리스를 잘하기 위해서는 쌓인 블록의 높이가 너무 한 부분으로 치우치지 않게 해야 한다. 또 쌓을 블록의 면이 기존에 쌓인 블록들이나 벽에 최대한 밀착하도록 쌓아 블록 사이에 공간이 최대한 생기지 않도록 해야 한다. 이런 방식을 기반으로 가중치를 선정하고 가중치에 따라 유전 알고리즘을 적용하여 최적의 플레이 방식을 찾도록 한다.

유전 알고리즘은 자연 세계의 진화 과정에 기초한 계산 모델로서 생물의 진화를 모방한 진화 연산의 대표적인 기법이다. 다윈의 적자생존 이론을 기본 개념으로 하는데 초기에 유전자(주어진 문제에 대해 가능한 해)들을 생성한다. 이 유전자들이 모여서 세대를 형성하고, 세대에서 가장 뛰어난 유전자들을 골라서 교배시킨다. 여기서 교배시킨다는 말의 의미는 유전자들은 각각 가중치들을 지

니고 있는데 뛰어난 유전자의 가중치들을 조합시켜 새로운 유전자를 생성한다는 것이다. 뛰어난 유전자들은 당연히 좋은 가중치들을 갖고 있을 확률이 높다. 알고리즘은 조건을 만족시키는 최적의 유전자가 1개 이상 나올 때까지 위의 단계들을 반복하고 조건을 만족하는 최적의 유전자 또는 유전자들이 나오면 종료된다.

게임 내에서도 마찬가지로 유전자들을 생성하고, 각 유전자는 각자가 가지고 있는 가중치에 따라 게임을 플레이한다. 여기서 유전자는 게임이 끝날 때까지의 플레이 1회를 말한다. 모든 유전자가 게임이 종료될 때까지 플레이한 후에 점수가 가장 높은 유전자들을 선택한다. 선택된 유전자들의 가중치들을 조합시켜 새로운 유전자들을 만든다. 앞의 단계들을 반복하여 실행하다가 조건을 만족시키면 알고리즘이 종료된다. 이때 살아남은 유전자들은 최적의(본문에서는 적어도 24시간동안 끝나지 않고) 게임 플레이를 하는 유전자들이다. 뒤에서 더 자세히 설명하도록 한다.

1.4 논문 구성

2장에서는 유전 알고리즘을 설명하고, 3장에서는 테트리스 플레이 방법과 플레이 방법을 기반으로 가중치를 선정하는 법과 각각의 가중치에 대하여 설명한다. 또 유전 알고리즘을 게임에 어떻게 적용했는지에 대해 설명한다. 4장에서는 게임 플레이가 가중치 개수와 연산에 따라 어떻게 변화하는지 실험하고, 게임 플레이의 성능을 평가한다. 5장에서는 결론으로 마무리한다.

2. 유전 알고리즘

2.1 유전 알고리즘 개요

유전 알고리즘은 자연계의 생물 유전학에 기본 이론을 두며, 병렬적이고 전역적인 탐색 알고리즘으로서, 다윈의 적자생존 이론을 기본 개념으로 한다. 유전 알고리즘은 주어진 문제에 대한 가능한 해들을 나열한다. 나열된 해들은 정해진 형태의 자료구조로 표현되는데 이들을 점진적으로 변형함으로써 점점 더 좋은 해들을 만들어 낸다. 여기에서 해들을 나타내는 자료구조는 유전자, 이들을 변형함으로써 점점 더 좋은 해를 만들어 내는 과정은 진화로 표현할 수 있다.

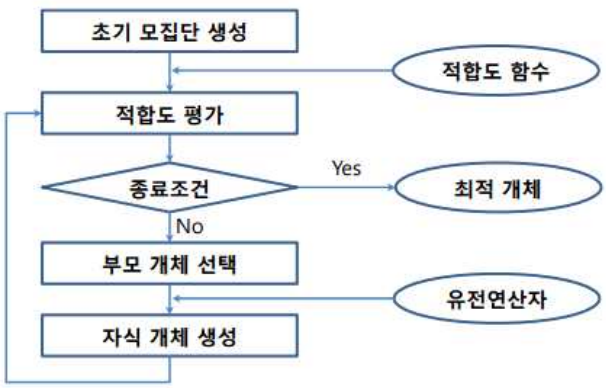
유전 알고리즘은 특정한 문제를 풀기 위한 알고리즘은 아니다. 문제를 풀기 위한 접근방법에 가까우며, 유전 알고리즘에서 사용할 수 있는 형식으로 바꾸어 표현할 수 있는 모든 문제에 대해서 적용할 수 있다. 또 유전 알고리즘은 자연 선택에 의해 해가 선택되므로 사용자가 해를 구하기 위해 복잡한 계산을 하지 않아도 된다. 일반적으로 문제가 계산 불가능할 정도로 지나치게 복잡할 경우 유전 알고리즘을 통하여, 실제 최적해를 구하지는 못하더라도 최적해에 가까운 답을 얻기 위한 방안으로써 접근할 수 있다. 이 경우 해당 문제를 푸는 데 최적화되어 있는 알고리즘보다 좋은 성능을 보여주지는 못하지만, 대부분 받아들일 수 있는 수준의 해를 보여줄 수 있다.

이러한 생물의 진화 과정, 즉 자연 선택과 유전 법칙 등을 모방한 알고리즘들로 진화 전략(Evolutionary strategies), 유전 프로그래밍(Genetic programming) 등 여러 형태의 이론과 기법들이 최근에 활발히 연구되고 있다. 유전 알고리즘은 이 중에서 가장 기본이 되고 대표적인 알고리즘으로, 자연과학/공학 및 인문 사회과학 분야에서 비선형 또는 계산 불가능한 복잡한 문제를 해결하는 데 널리 응용되고 있다.

2.2 유전 알고리즘의 구성

유전 알고리즘은 유전자들의 집합으로부터 적합도가 가장 좋은 유전자를 선택하고, 선택된 해의 방향으로 탐색을 반복하며 최적 해를 찾아가는 구조로 동작한다. 유전 알고리즘의 동작을 단계별로 표현하면 아래와 같다.

(1) 초기 유전자들의 집합 생성
(2) 유전자들에 대한 적합도 계산
(3) 종료 조건 판별
    1) 종료 조건이 참인 경우, 알고리즘 종료
    2) 종료 조건이 거짓인 경우, 4)로 이동
(4) 현재 집합에서 가장 좋은 유전자들 선택 (부모 선택)
(5) 선택된 유전자들을 조합하여 새로운 유전자들 생성 (자손 생성)
(6) 3)으로 이동하여 반복



그림[2-1] 유전 알고리즘 흐름도

유전 알고리즘은 초기 모집단을 생성한 후에 적합도 함수로 각각의 해가 적합한지에 대해 평가한다. 평가 후엔 종료 조건을 판별하여 만족할만한 수준의 해가 나왔는지 확인한다. 해가 조건에 부합하면 알고리즘을 종료하고, 조건에 부합하지 않으면 알고리즘을 계속 진행한다. 조건에 부합하지 못 하여 알고리즘이 계속 진행된다면 현재 집합에서 가장 좋은 유전자(해)들을 선택하고, 선택된 유전자들을 연산하여 새로운 유전자 집단을 만든다. 초기 모집단이 1세대였다면 새로 만들어진 집단은 2세대이다. 조건을 만족하여 알고리즘이 종료될 때까지 세대를 거듭하여 좋은 유전자를 선택하는 것이 유전 알고리즘이다.

유전 알고리즘을 어떤 문제 적용하기 위해서는 총 4개의 아래와 같은 연산이 필요하다.

1) 초기 염색체를 생성하는 연산
2) 적합도를 계산하는 연산
3) 적합도를 기준으로 유전자를 선택하는 연산
4) 선택된 유전자들로부터 자손을 생성하는 연산

각각의 연산에 대해서는 아래에 자세히 설명한다. 그러나 아래의 설명은 유전 알고리즘의 일반적인 예시일 뿐이며 문제나 필요에 따라서 연산을 수정할 수 있다.

알고리즘이 처음 시작될 때는 이전 유전자 집단이 존재하지 않기 때문에 우수한 유전자를 선택하여 자손을 생성할 수 없다. 따라서 초기 유전자 집단을 생성하는 연산을 별도로 정의해야 한다. 가장 많이 이용되는 방법은 어떠한 규칙도 없이 임의의 값으로 유전자를 생성하는 것이다. 자연 선택에 의해 좋지 않은 유전자는 도태되므로 임의의 값으로 유전자를 생성해주어도 알고리즘엔 큰 영향이 없다.

유전 알고리즘 적용에 필요한 적합도를 계산하는 연산은 유전자에 표현된 정보를 기반으로 적합도를 계산하는 연산이다. 이 연산은 해결하고자 하는 문제에 매우 종속적이므로 3장에서 자세히 설명한다.

선택 연산은 다음 세대의 유전자 집합을 생성하기 위해 부모 유전자를 선택하

기 위한 연산이다. 유전자를 선택하는 연산은 유전 알고리즘에서 가장 중요한 연산 중 하나인데 어떤 연산을 적용하는지에 따라 알고리즘의 성능이 달라질 수 있다. 선택 연산의 종류로는 품질 비례 룰렛 휠 선택, 토너먼트 선택, 순위기반 선택 등 다양한 연산들이 있으나 공통된 원칙은 우수한 해가 선택될 확률이 높아야 한다는 것이다. 우수한 해들과 열등한 새들 사이의 적합도 차이를 조절함으로써 선택 확률을 조절할 수 있는데, 이 차이의 정도를 선택압(selection pressure)이라 한다. 선택압이 높을수록 해가 한 쪽으로 빠르게 수렴될 수 있으나 최적의 수렴이 아닌 나쁜 수렴의 가능성도 높아진다. 즉, 유전적 다양성을 해치게 된다. 선택된 유전자들은 마치 생명체가 교배를 통해 유전자들을 섞듯, 일정한 규칙에 따라 서로 섞이는데 선택압을 너무 높게 설정한다면 어느 범주에 치우쳐진 유전자의 자손만이 살아남아 유전적 다양성을 형성하지 못하게 된다. 그렇게 되면 더 나은 발전을 하지 못하고 일정한 수준에서 진화를 멈춰버리는 현상이 발생한다. 이런 현상을 막기 위해서 유전자 다양성의 적정선을 지켜 당장은 좋지 않은 유전자일지라도 세대에 약간의 변동을 주는, 진화의 매개가 될 수 있는 유전자들을 살아남게끔 해야 한다. 반대로 선택압이 너무 낮으면 해 집단의 평균 품질이 좋아지지 않을 가능성이 있다. 아래에 몇 가지 대표적인 선택 연산자들을 소개한다.

첫 번째로 소개할 선택 연산은 가장 대표적인 선택 방법으로 품질 비례 룰렛 휠이다. 각 해의 품질을 평가한 다음 가장 좋은 해의 적합도가 가장 나쁜 해의 적합도보다 k배가 되도록 조절한다. 해 집단 내의 해 i의 적합도는 다음과 같이 계산된다.

$$f_i = (C_w - C_i) + (C_w - C_b)/(k-1) \quad (\mathrm{k} > 1)$$

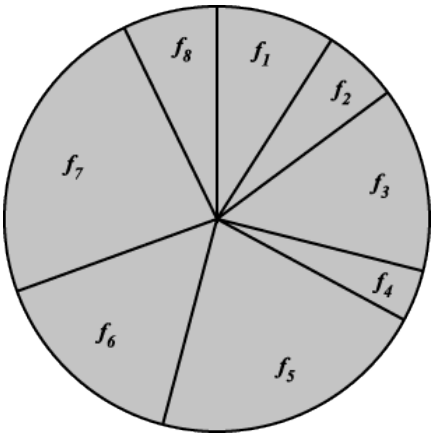
C_w : 해 집단 내에서 가장 나쁜 해의 비용

C_b : 해 집단 내에서 가장 좋은 해의 비용

C_i : 해 I의 비용

위의 계산으로 각 유전자의 적합도를 구한 후에 유전자의 적합도의 비로 전체

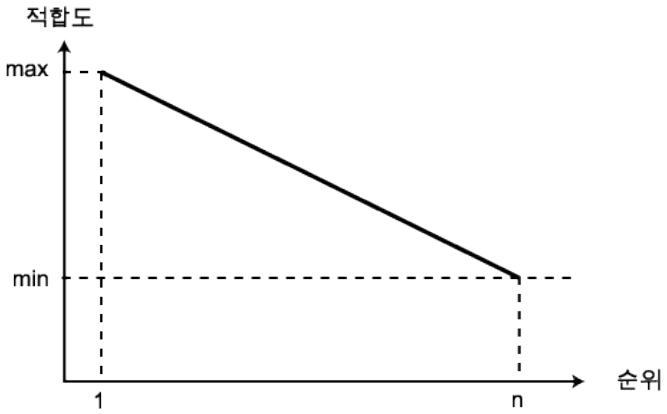
세대를 비례 배분하여 룰렛 휠에서 각 유전자가 차지하는 지분을 결정하는데, 적합도 만큼의 공간을 배정한다. 그런 후엔 룰렛 휠의 범위 내에서 난수를 생성하여 그 난수 범위에 해당하는 유전자를 선택한다. 난수를 발생하여 선택하게 되면 각 유전자의 선택 확률은 배정된 공간의 크기에 비례하게 된다.



그림[2-2] 룰렛 휠 공간 배정 예

다음 소개할 연산은 토너먼트 선택이다. 가장 간단한 선택 연산으로 세대에서 두 개의 유전자를 임의로 선택하고, 난수 r을 생성한다. 이 난수를 미리 정의한 선택압 k와 대소를 따져 난수 r이 k보다 작다면 두 유전자 중 품질이 좋은 것을 선택하고, 난수가 k보다 크다면 품질이 나쁜 것을 선택한다. 난수를 생성할 때는 일정 범위 내에서 생성되도록 설정하고, 선택압인 k는 난수 범위의 중간 값 보다는 크게 설정해야 한다. 만약 k가 난수 범위의 중간 값 보다 작다면 품질이 나쁜 것을 선택할 확률이 커져 유전 알고리즘의 바탕 이론인 자연선택에 비합리적이다. 반면에 k가 높아질수록 선택압이 커지기 때문에 k값을 적당한 값으로 설정하여 적합도가 낮은 유전자들도 선택될 수 있게 해야 한다. 또는 유전자 풀에서 유전자 몇 개를 선택하여 이들을 토너먼트 형식으로 비교하여 마지막에 남는 것을 선택하는 방법도 있다.

앞에서 설명한 품질 비례 룰렛 휠 선택에서는 k값을 조정함으로써 좋은 품질의 해와 나쁜 품질의 해가 지나치게 적합도의 차이를 갖게 되는 것은 막을 수 있었다. 그러나 해들의 적합도 분포는 조절할 수 없다. 순위 기반 선택은 이것을 보완하는데 해 집단 내의 해들을 품질 순으로 순위를 매긴 다음 가장 좋은 해부터 일차 함수적으로 적합도를 재정하는 방법이다. 그림 [2-3]은 순위 기반 선택의 적합도 배정 함수를 그림으로 보여준다.



그림[2-3] 순위 기반 선택의 적합도 배정 함수

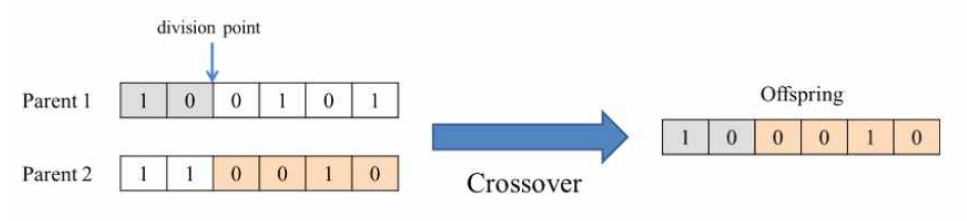
n개의 유전자 중 i번째 순위를 가진 염색체의 적합도는 다음과 같은 식으로 계산할 수 있다.

$$f_i = \max + (i-1) \times (\min - \max)/(n-1)$$

이 식에서 max와 min값의 차이를 바꿈으로써 선택압을 조절할 수 있다. 해들의 적합도는 max와 min 사이에서 균일한 간격으로 분포하게 된다.

유전자 알고리즘의 중요한 연산 가운데 다른 하나는 자손을 생성하는 연산이다. 크게 교차, 변이, 대치 연산이 있고, 이 세 가지 연산들은 또 세부적으로 여러 가지 방법들을 가지고 있는데 본 논문에서는 교차와 변이가 무엇인지에 대해서만

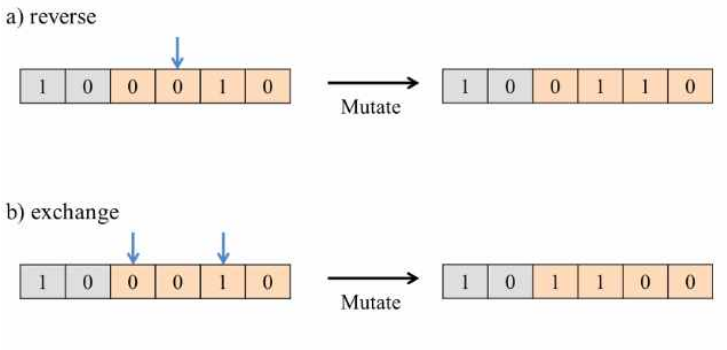
간략하게 소개한다.



그림[2-4] 교차 연산의 동작

자손은 2개의 유전자 부모로부터 생성되는데 교차 연산에서는 분할하는 지점인 division point를 임의로 선택하여 한 부모의 분할 지점 앞부분과 다른 부모의 분할지점 뒷부분을 합하여 자손을 생성한다. 위의 그림엔 division point가 하나인데 이것을 일점 교차라고 한다. 길이가 n인 일차원 문자열로 된 유전자 상에서 일점 교차로 자르는 방법의 총 수는 n-1가지이다.

교차 연산만을 이용하여 세대를 진화시킨다면 이 세대는 지역 최적점에 도달하게 될 것이다. 이 문제를 해결하기 위하여 유전 알고리즘에서는 새롭게 생성된 유전자에 확률적으로 돌연변이가 발생하도록 한다. 생성된 돌연변이는 임의로 생성해주는 것이기 때문에 품질이 나쁜 유전자가 생성될 확률이 있다. 때문에 일반적으로 한 세대에서 아주 낮은 확률로 예를 들면, 2% 미만의 유전자들만 돌연변이가 발생하도록 설정하여 지역 최적점에 빠지지 않도록 한다. 아래 그림은 돌연변이 연산에 대한 두 가지 예시를 보여준다.



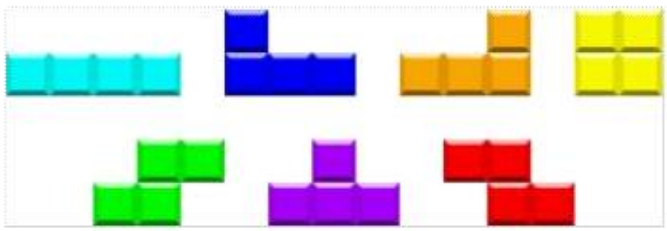
그림[2-5] 돌연변이 연산에 대한 두 가지 예시

그림 [2-5]에서 a)는 임의로 하나의 유전자를 선택하여 0이면 1로, 1이면 0으로 값을 바꾸는 돌연변이 연산이다. b)는 두 개의 유전자를 임의로 선택하여 두 유전자의 값을 교환하는 돌연변이 연산이다. 돌연변이 연산은 이외에도 많은 종류가 있으며, 해결하고자 하는 문제에 맞게 돌연변이 연산을 정의하면 된다.

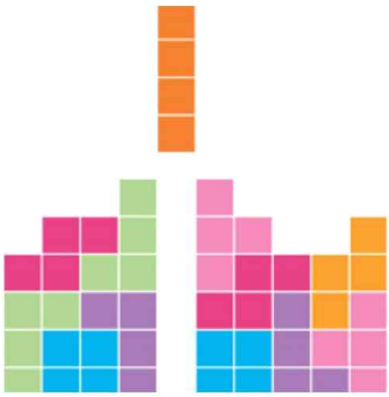
3. 유전 알고리즘을 이용한 테트리스 플레이

3.1 테트리스 플레이

테트리스는 퍼즐 게임으로 네 개의 사각형으로 이루어진 서로 다른 모양의 블록들을 차곡차곡 쌓아 최대한 오랫동안 살아남는 게임이다. 그림 3-1에 보이는 7개의 서로 다른 모양의 블록을 테트로미노라고 한다. 이 7개의 블록을 90도씩 회전시키거나 좌우로 움직여서 가로 10칸, 세로 20칸의 맵에 차곡차곡 쌓는데, 줄(한 행, 사각형 10개 크기)이 블록들로 빈틈없이 가득 차면 그 줄은 제거된다. 줄이 제거되면 그 위에 쌓인 다른 블록들이 제거된 줄 개수만큼 아래로 내려온다. 결국, 오래 살아남으려면 빈틈없이 쌓아서 줄을 계속 없애주어야 한다.

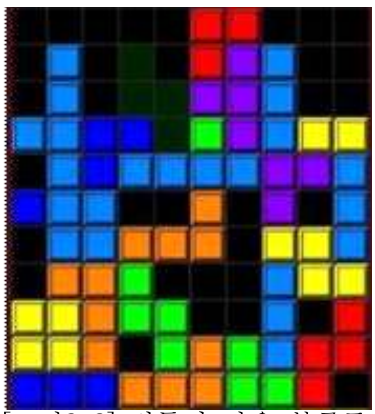


[그림3-1] 7개의 테트로미노

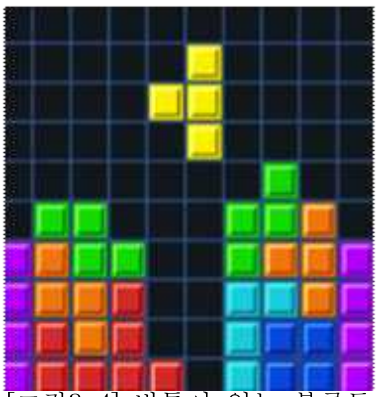


[그림3-2] 한 줄이 빈틈없이 채워지는 모습

본 논문은 최적의 테트리스 AI 구현을 목표로 한다. 그렇다면 이것을 위해 테트리스를 잘하기 위한 플레이 방식에 대해 고민해볼 필요가 있다. 먼저 블록의 높이가 한 부분에 너무 치우치지 말아야 한다. 한 부분에 치우치게 되면, 다음 블록의 위치를 결정할 때 이미 높이 쌓아져 있는 부분에 더 쌓기가 부담스럽기 때문에(게임이 끝날 수 있다) 놓을 수 있는 위치가 한정된다. 또 블록 사이에 공간이 최대한 생기지 않도록 해야 한다. 블록 사이에 공간이 있다는 것은 한 줄(가로 행)을 완전히 채우지 못했으므로 당연하게도 그 줄이 당분간은 지워지지 않는다. 앞서 말한 두 가지의 방식을 생각해보면, 우리가 블록을 놓을 때 최대한 다른 블록이나 벽에 밀착하는 것이 좋다. 밀착하지 않으면 그만큼 공간이 생길 가능성이 크고 높이가 더 높아질 가능성이 크다.



[그림3-3] 빈틈이 많은 블록들



[그림3-4] 빈틈이 없는 블록들

3.2 가중치

유전 알고리즘을 어떤 문제에 적용하기 위해서는 해를 유전자 형식으로 표현하는 것이 필요하다. 따라서 테트리스에 유전 알고리즘을 적용하기 위해서는 플레이에 중요한 영향을 끼치는 요소들을 유전자 형식으로 표현하는 것이 필요하다. 앞에서 최적의 테트리스 AI 구현을 위해 테트리스 플레이 방법에 대해 생각해보았고, 본 연구에서는 이 요소들을 가중치로 설정하여 유전 알고리즘을 적용하였다.

만약 가중치를 하나로 설정한다면 최적화를 할 수 없을 것이다. 왜냐하면, 블록의 다음 위치를 결정할 때 가중치의 개수가 1개라면 가중치를 어떻게 설정하든지 간에 블록을 놓는 위치를 전부 반영할 수 없을 것이다. 따라서 적절한 수의 가중치의 개수를 설정해줘야 한다.

우리가 플레이어라고 생각하고, 하나의 블록이 내려올 때 블록을 놓을 위치를 결정한다고 생각해보자. [그림3-5]를 [그림3-6]에 놓는다고 생각해보면, 높이의 편차를 줄이기 위함이라면 맨 오른쪽에 쌓을 수 있다. 그러나 블록끼리 맞닿는 면을 최대로 하려면 중간에 위치할 수 있을 것이다. 이렇게 하나의 가중치로 해결할 수 없는 부분이 있다. 따라서 문제에서 중요한 요소를 고려하여 가중치로 설정해주는 것이 필요하다.

[그림3-5] 현재 내려오는 블록

[그림3-6] 쌓아 올린 블록들

1. 쌓인 블록의 최대 높이
2. 높이 편차
3. 블록을 놓았을 때 생기는 공간의 수
4. 블록을 놓았을 때 완성되는 줄의 수
5. 현재 블록이 기존의 블록이나 바닥, 벽에 닿는 면의 수 (인접한 면의 수)

테트리스 플레이에서 중요한 요소들은 위와 같다. 블록을 잘 쌓기 위해서는 블록의 최대 높이가 낮고, 높이 편차가 작으며, 블록을 놓았을 때 생기는 공간의 수가 적고, 블록을 놓았을 때 완성되는 줄 수가 많아야 한다. 또 대부분은 블록이 놓였을 때 인접한 면의 수가 많을수록 좋다. 본문에선 최종적으로 위의 요소 중 4개를 가중치로 설정하여 블록이 놓일 최적의 위치를 선택하게 하였다. 선정된 4개의 가중치는 1번부터 4번까지의 요소들이다.

간단히 정리하자면 가중치란 플레이에 영향을 주는 요소들이고, 블록들은 설정된 가중치에 따라 놓이게 된다. 아래 그림에 가중치에 대한 간단한 예시가 있다. [그림3-7]을 보면 각각 가중치는 최대 높이 14, 높이 편차 6.5, 공간의 수 7, 완성되는 줄의 수 0, 인접한 면의 수 4라는 것을 알 수 있다. 가중치에 따라 블록을 쌓는 것에 대한 부분은 뒤에서 자세하게 설명하기로 한다.

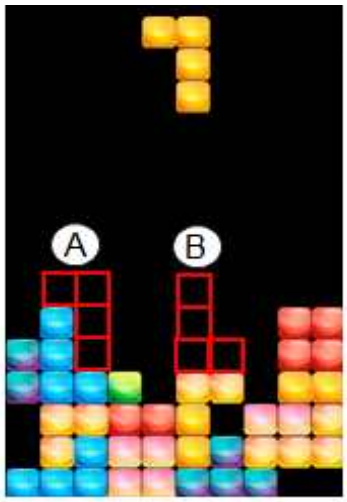
[그림3-7] 5개의 가중치 요소들

3.3 유전 알고리즘 적용

위에서 설정한 가중치들을 유전 알고리즘에 적용하여 표현하면 다음과 같다.

gene(i) = weight(1) + weight(2) + weight(3) + ... + weight(k)
gene pool = {gene(1), gene(2), gene(3), ... gene(n)}

그러면 유전자를 이루고 있는 가중치들은 실제로 어떻게 작동할까?
앞에서 설명한 가중치 항목들을 w_1부터 w_k까지 나타낸다고 했을 때, 이 가중치들은 가중치 계수를 가지며, 초기 가중치 계수는 무작위로 정해진다.
즉, 초기 시작은 유전 알고리즘의 특성인 자연 선택을 계승하여 사용자가 특정 값을 입력하는 것이 아니라 무작위 값으로 정하게 한다.
예를 들어서, 유전자 1번의 가중치가 4개 있다고 하자. 초기 값으로 1번 가중치에 0.4, 2번 가중치에 0.2, 3번 가중치에 -0.5, 4번 가중치에 0.0 이런 식으로 할당된다. 이것을 간단하게 G1 = {0.4, 0.2, -0.5, 0.0}과 같이 표현할 수 있다.
하나의 유전자가 게임을 플레이할 때, 자신이 부여받은 가중치 계수와 현재 블록의 위치에 따른 실제 값을 곱한 것이 가중치 값이 된다. 여기서 실제 값이라는 것은 블록이 어떤 위치에 놓였을 때, 각 가중치 계수에 곱해지는 상수를 말한다. 예를 들어, 그림 [3-8]의 상황에서 가중치(인접한 면의 수) 하나만 있다고 가정하자. 이 가중치는 블록이 놓였을 때 인접한 면의 수를 실제 값으로 가진다. A는 4이고, B는 2의 값을 가진다. 이때, 가중치의 계수가 0.3이라고 한다면 A 위치의 가중치 값은 0.3 x 4 = 1.2이고, B 위치의 가중치 값은 0.3 x 2 = 0.6이 된다. B 위치보다 A 위치에 놓았을 때 가중치 합이 더 높으므로 최종적으로 블록은 A 위치에 놓이게 된다.



[그림3-8] 가중치 값 계산

앞에선 가중치 하나를 가지고 예를 들었지만 실제로 한 유전자(개체)는 여러 개의 가중치를 갖고 있다. 본 연구에선 이러한 여러 개의 가중치를 고려하여 최적의 블록 위치를 선정하기 위해 적합도 함수를 사용한다.

적합도 함수란 블록이 어떤 위치에 놓였을 때 계산된 가중치들의 총합이다. 앞에서 설명한 것처럼 가중치 값들은 (실제 값 x 가중치 계수)로 이뤄져 있고, 모든 가중치 값들을 더한 것이 적합도 함수이다. 블록을 놓을 수 있는 모든 위치에 각각의 가중치들의 총합인 적함도 함수를 구한 후에 서로의 가중치 합을 비교한다. 이때 가중치의 값이 제일 큰 곳이 적합하며 최종적으로 블록이 놓일 위치가 된다. 예를 들면, 블록이 놓일 수 있는 위치가 A부터 E까지 다섯 곳이 있다고 가정하자. A 위치의 적합도 함수의 값은 100, B 위치의 값은 90, C 위치의 값은 80, D 위치의 값은 70, E 위치의 값은 60이라고 할 때, A의 가중치 총합(적합도)이 가장 크므로 블록은 A 위치에 놓인다.

* 적합도 평가 단계

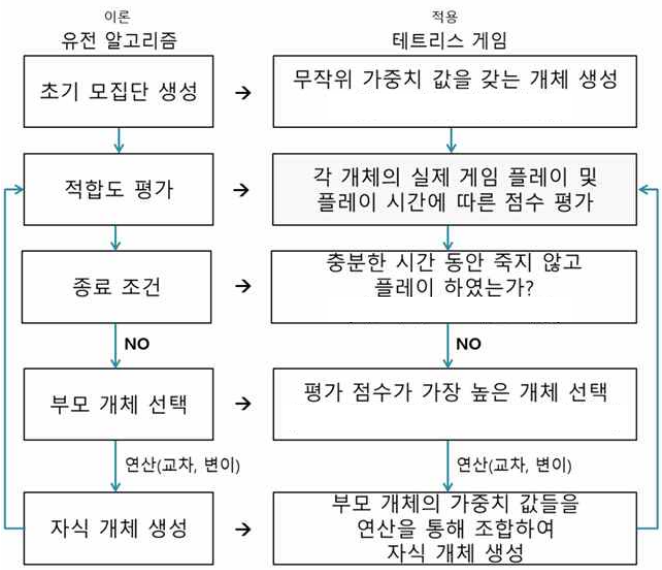
(1) 각각의 가중치 계수와 상수를 곱한다. weight = c x k

(2) 모든 가중치를 합한다. ∑weight

(3) 가중치의 합이 제일 크게 하는 가중치들을 적합도 함수의 가중치로 결정한다. max(f1:∑weight, f2:∑weight, ... , fn:∑weight)

이제까지는 유전 알고리즘을 테트리스에 적용하기 위해 문제의 해를 유전자 형식으로 표현하는 것과 유전자(가중치)에 따라 테트리스가 어떤 방식으로 플레이되는지 설명하였다.

지금부터는 알고리즘의 전체적인 흐름을 살펴보자. 아래의 그림 [3-9]는 테트리스에 적용한 유전 알고리즘의 전체적인 흐름을 보여준다.



[그림 3-9] 유전 알고리즘 흐름도

초기 세대(0세대)가 생성될 때, 무작위 가중치 값을 갖는 유전자(개체)들 100개가 생성된다. 각각의 유전자는 앞에서 설명한 방식대로 가지고 있는 가중치에 따라 테트리스를 플레이한다. 유전자 풀에 있는 모든 유전자들의 플레이가 끝난 후엔 선택 연산을 통하여 부모 개체를 선택한다. 선택된 부모 개체들은 교차 연산을 통하여 자식 개체를 생성하고 다음 세대로 가게 된다. 알고리즘은 이 문제의 최적 해인 최적의 가중치를 갖는 세대를 찾을 때까지 위의 과정을 반복한다.

본문에선 선택 연산 중에 토너먼트 방식을 활용하였다. 유전자 풀에서 10%에 해당하는 유전자들을 임의로 선택하고 이 중에서 가장 적합도가 높은 두

유전자를 선택하여 교차 연산한다. 어느 유전자가 적합한지는 가중치에 따른 플레이를 기반으로 점수를 매겨 판단한다. 여기서 점수는 제거된 줄(행)의 수이다. 새로 생성되는 개체가 전체 유전자 풀의 30%가 될 때까지 이 과정을 반복한다. 다음 세대는 기존 세대의 70% 개체와 새로 생긴 30%의 개체로 구성된다.

다음 세대의 새로운 개체를 생성하기 위한 교차 연산에는 가중 평균 방식을 사용하였다. 가중 평균 방식은 부모 개체로 A, B가 선택되었을 때, 각 개체의 적합도에 따라 편향을 갖는 형태로 자식 개체를 생성한다. 이렇게 하면 교차 연산하려는 두 개체 중 더 큰 적합도를 갖는 개체 쪽으로 편향된 새로운 가중치를 갖는 유전자를 만들 수 있다. 초기에는 우수한 개체들의 가중치들을 바꿔주는 방식으로 진행하였으나, 돌연변이 연산 외에는 기존에 생성된 가중치들을 바꾸지 못하고 몇 세대 지나지 않아서 지역 최대 문제에 빠지게 되었다. 따라서 가중 평균 방식을 사용하여 교차 연산될 때마다 완전히 새로운 가중치들을 갖는 개체들을 생성하여 이 문제를 해결하였다.

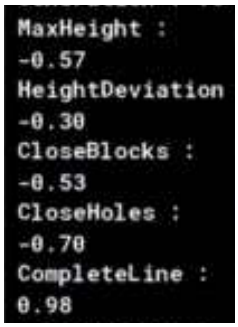
앞선 선택 연산에서 유전적 다양성을 증가시키기 위해 가장 상위의 개체만을 선택하는 것이 아닌 임의의 범위 내에서 가장 좋은 유전자를 선택하는 방식을 취했다. 이 방식으로 선택된 개체는 전체 세대에서 비교적 좋은 품질의 유전자가 아닐 수 있다. 하지만 가중 평균 방식을 이용한 교차 연산은 선택된 개체 중 더 좋은 품질의 유전자를 따라가게 된다. 이렇게 유전적 다양성을 지키면서 더 나은 개체를 생성할 수 있었다.

다음 세대 생성 과정 중에는 변이 연산도 사용한다. 변이 연산은 일정 확률로 한 개체의 가중치를 임의의 값으로 변경한다. 앞에서도 말했듯이 교차 연산만을 이용하여 세대를 진화시킨다면 지역 최대 문제를 발생시킬 수 있기 때문에 이를 해결하기 위하여 돌연변이 연산을 사용하였다. 하지만 돌연변이 연산은 임의의 값을 생성하므로 한 세대에서 돌연변이의 발생 비율이 너무 높으면 발전에 방해가 되므로 한 개체당 5%의 확률로 돌연변이가 발생하도록 설정하였다.

4. 성능 평가

본 연구의 목표는 테트리스 게임 내에서 최적 해인 최적의 플레이를 찾는 것이다. 컴퓨터가 알고리즘에 따라 게임을 하고, 게임이 끝났을 때 플레이를 평가한다. 최소 24시간 이상 살아남고 플레이를 하였는지 검사한다. 테트리스 게임에서 끝나지 않고 계속해서 플레이하기 위해 알고리즘에서 가장 중요한 요소를 꼽자면 플레이를 결정하는 가중치라고 말할 수 있다. 테트리스 게임 내에서 블록들이 놓일 위치는 가중치에 따라 결정된다. 앞에서 설명했듯 가중치 평가는 블록이 놓였을 때 그 위치의 실제 값과 유전자가 가지고 있는 가중치 계수를 곱하여 가장 가중치가 높은 곳을 선택하여 그 위치에 블록을 놓는다. 그렇기에 가중치 계수의 값이 중요하다. 좋은 가중치를 가지고 있을수록 더 좋은 플레이를 할 확률이 올라간다.

여기서 좋은 가중치란 값이 커지면, 게임에 긍정적인 영향을 주고 반대로 값이 작아지면 부정적인 영향을 준다고 판단할 수 있는 단계이다. 따라서 좋은 플레이를 하는 유전자들은 이 가중치 계수 값에 어느 정도 수렴하는 값들을 갖고 있다. 예를 들어, 블록의 최대 높이, 높이의 편차, 공간을 막고 있는 블록의 개수, 막힌 공간의 개수, 블록을 놓았을 때 완성될 줄의 개수를 가중치로 설정했다고 가정하자. 테트리스 플레이에서 블록의 최대 높이는 낮을수록 안전하다. 그렇기에 해당 가중치의 값이 작을수록 좋은 플레이를 하게 된다. 따라서 좋은 플레이를 하는 유전자들은 이 가중치 계수의 값이 음수 값을 갖는다. 마찬가지로 높이의 편차가 낮을수록 좋은 플레이를 한다고 말할 수 있고, 또한 공간을 막고 있는 블록과 블록을 놓았을 때 막혀 있는 공간의 개수가 적을수록 좋다. 그래서 좋은 플레이를 하는 유전자들은 이 4개의 가중치 계수에 대해 음수 값을 갖거나 양수더라도 낮은 가중치 계수 값을 갖는다. 반대로 블록을 놓았을 때 완성되는 줄의 개수는 많을수록 좋다. 따라서 이 가중치 계수의 값이 더 큰 양수 값을 가질수록 더 좋은 플레이를 할 가능성이 커진다. 그렇기 때문에 좋은 플레이를 하는 유전자들은 해당 가중치 계수 값이 양수 값을 갖는다.

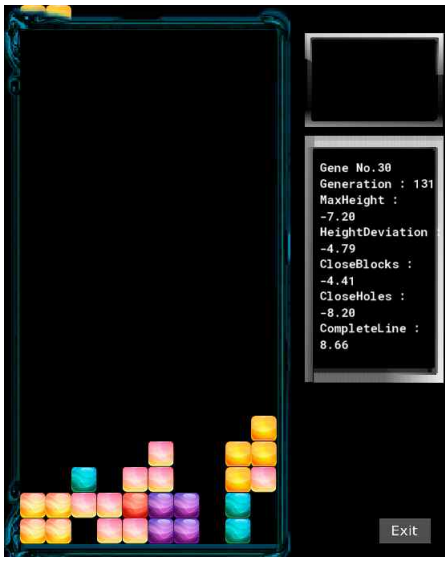


그림[4-1] 가중치 계수들의 값

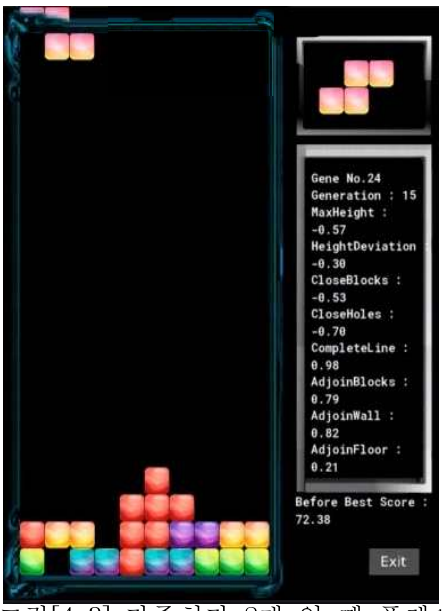
[그림4-1]은 좋은 플레이를 하는 유전자의 가중치 계수를 보여준다. MaxHeight는 최대 높이를 나타내는 가중치이고, HeightDeviation은 블록들의 높이의 편차, CloseBlocks은 공간을 막고 있는 블록의 수, CloseHoles은 막힌 공간의 수, CompleteLine은 완성된 줄의 수를 뜻한다. 앞에서 설명했듯 위의 4개는 마이너스 값을 가지고 있고, 아래 1개는 플러스 값을 가지고 있다. 좋은 플레이를 하는 즉, 성능이 좋은 모든 유전자는 이와 같은 공통점을 가지고 있다.

이렇듯 기대되는 가중치 계수 값을 가지고 있는 유전자가 더 좋은 플레이를 하게 되기 때문에 가중치 계수의 값이 중요하고, 알고리즘의 핵심은 좋은 가중치 계수 값을 찾아가는 것이다. 그런데 가중치 계수의 값이 좋은 것보다 근본적으로 생각해볼 것이 있다. 바로 유전자가 가지는 가중치의 총 개수이다.

연구 초기엔 총 5개의 가중치들을 가지도록 설정하였다. 블록의 최대 높이, 높이의 편차, 공간을 막고 있는 블록의 개수, 막힌 공간의 개수, 블록을 놓았을 때 완성될 줄의 개수를 가중치들로 두고 게임을 하도록 하였다. 이렇게 5개의 가중치들을 가지고 게임을 하였을 때 유전자들은 일정 세대까지만 발전하고 그 이후로는 130세대가 지나도 발전하지 않는 것을 발견하였다. 이에 필자는 게임에 영향을 주는 요소들인 가중치들의 개수가 적기 때문이라는 가설을 세우고 게임에 영향을 주는 요소들을 추가로 가중치로 설정하였다. 추가한 가중치들은 총 3개로 현재 블록이 놓였을 때 벽과 닿는 블록면의 수, 현재 블록이 놓였을 때 바닥과 닿는 블록면의 수, 현재 블록이 놓였을 때 기존의 블록과 닿는 블록면의 수를 추가로 설정하였다. 아래의 그림들은 각각 5개의 가중치를 가질 때와 8개의 가중치를 가질 때를 캡처한 것이다.



그림[4-2] 가중치가 5개 일 때 플레이



그림[4-3] 가중치가 8개 일 때 플레이

총 8개의 가중치를 가지고 게임을 하였을 때 불과 10세대 만에 기존 5개의 가중치를 가지고 게임을 하였을 때보다 더 나은 발전을 하는 것을 확인하였다. 하지만 15세대 이후 대부분의 유전자들이 비슷한 가중치를 가지고 더 이상 발전하지 않는 것을 확인하였다. 가중치를 추가하기 전보다는 더 나은 모습을 보였지만 최적의 해라고 생각하는 끊임없이 플레이한다는 조건을 만족하지 못 하고 지역 최적점에 빠지는 것을 확인하였다. 이에 가중치의 개수뿐만 아니라 발전에 영향을 끼치는 다른 중요한 요소에 대해 생각해보게 되었다. 하지만 그 전에 가중치들의 개수가 플레이에 어떤 영향을 미치는지 확인하기 위해 기존의 5개의 가중치를 가진 유전자와 8개의 가중치를 가진 유전자의 플레이를 비교하였다. 비교 결과 8개의 가중치를 가진 유전자가 5개의 가중치를 가진 유전자보다 더욱 빠르게 발전하지만, 일정 세대가 지나면 오히려 기존의 플레이와 거의 차이가 없는 플레이를 한다는 것을 발견하였다.

유전자는 가중치에 따라 플레이를 하게 되는데 이는 가중치가 플레이를 제약하는 요소라는 뜻도 된다. 따라서 가중치가 많아졌다는 말은 제약이 많아졌다는 의미도 된다. 꼭 필요한 요소를 가중치로 넣는 것은 좋은 플레이를 기대할 수 있지만, 중복되거나 중요하지 않은 요소를 가중치로 넣는 것은 오히려 발전에 방해가 된다. 따라서 필자는 추가한 가중치들이 영향이 없을 수 있다고 생각하고 가중치들에 대해 유심히 생각해보았다. 추가된 3개의 가중치들은 현재 블록이 놓였을 때 벽과 닿는 블록면의 수, 현재 블록이 놓였을 때 바닥과 닿는 블록면의 수, 현재 블록이 놓였을 때 기존의 블록과 닿는 블록면의 수이다. 이 3가지 요소는 테트리스에 영향을 미치는 중요한 요소이다. 블록이 기존의 블록이나 벽, 바닥에 접하는 면이 많을수록 공간을 만들지 않으며 블록을 쌓을 수 있다. 하지만 이것은 이미 있는 가중치 요소인 블록이 놓였을 때 생기는 공간의 개수에 포함되는 부분이 있다. 이미 포함된 요소인데 따로 가중치로 만들었기 때문에 플레이에 제약을 주는 요소가 되었고, 좋지 못한 플레이를 하게 한 원인이 되었다.

이것을 보면 유전자들이 가지는 가중치의 총 개수가 알고리즘 성능에 큰 영향을 주지만 적절하지 못한 요소를 가중치로 두었을 때는 오히려 제약으로 작용할 수 있다는 것을 알 수 있었다. 가중치의 개수가 많을수록 좋은 쪽으로 발전할 수

있지만 무작정 가중치의 개수를 늘려서는 안 된다. 그 가중치 항목이 과연 현재 문제에 대해 적절한지 충분히 생각해보아야 한다. 본 논문에서 가중치 항목은 최적의 테트리스 플레이를 위한 것이므로 가중치는 테트리스의 좋은 플레이와 연관되어야 한다. 적절한 가중치 항목의 추가는 평가할 요소가 더욱 많아져 이전보다 더 세세하게 평가할 수 있게 하고, 더욱 빠르게 좋은 방향으로 진화할 수 있게 한다. 하지만 적절하지 못한 가중치 항목은 오히려 발전을 저하시키기 때문에 무엇을 가중치로 선정할지가 유전 알고리즘에서 가장 중요한 것들 중 하나이다. 위의 생각을 반영해 다른 중요한 요소를 포함하면서 꼭 필수적인 요소만 가중치로 선정하였다. 최대 높이, 높이 편차, 블록 사이의 공간의 수, 완성된 줄 수로 총 4개이다. 시뮬레이션 결과 기존의 결과와 비슷하거나 상회하는 모습을 보였다.

또한 중요한 것들 중 하나는 다음 세대를 생성하기 위해 부모를 선택하는 선택 연산과 그 부모들이 가진 가중치들을 어떻게 조합하여 다음 유전자들을 생성할지에 대한 교차, 변이 연산이다. 본 연구에선 초기에 플레이에 따라 점수를 매기고, 점수 순으로 정렬하여 상위의 몇 개 유전자들만을 선택하여 그 유전자들의 가중치들을 조합하는 방식으로 다음 세대를 만들었다. 그렇게 했을 경우, 계속 상위 유전자들만을 선택함으로 선택에 있어서 유전적 다양성의 범위가 좁아 유전자들이 세대를 얼마 지나지 않았음에도 같은 가중치 값을 가지게 되었다. 아래의 그림은 플레이가 끝난 후 각 세대의 상위 5개 유전자들의 적합도와 가중치들을 텍스트 파일에 기록한 것이다. 텍스트 1줄이 유전자 한 개의 적합도와 가중치를 나타낸다. 가장 앞에 있는 숫자가 적합도이고 그 뒤의 8개의 실수들이 가중치 계수를 나타낸다. 그림에서 7세대가 되었을 때, 유전자들의 가중치 계수 값들이 같아지게 되는 것을 볼 수 있다.

```
Generation : 0
12167 0.14 -0.52 0.03 -0.84 0.38 0.98 0.65 -0.58
823 -0.99 0.87 -0.59 -0.31 -0.82 0.88 0.04 0.89
513 0.44 0.97 0.12 -0.36 0.87 0.49 0.42 0.89
300 -0.82 -0.48 -0.12 -0.96 0.21 -0.98 0.52 0.24
290 -0.55 0.33 -0.45 0.09 0.13 0.14 -0.93 -0.10
```

그림[4-4] 가중치가 8개 일 때 플레이 (0세대)

```
Generation : 7
89756 -0.55 -0.52 -0.12 -0.96 0.21 0.49 0.52 0.89
78104 -0.55 -0.52 -0.12 -0.96 0.21 0.49 0.52 0.89
50012 -0.55 -0.52 -0.12 -0.96 0.21 0.49 0.52 0.89
45678 -0.55 -0.52 -0.12 -0.96 0.21 0.49 0.52 0.89
46789 -0.55 -0.52 -0.12 -0.96 0.21 0.49 0.52 0.89
```

그림[4-5] 가중치가 8개 일 때 플레이 (7세대)

이것을 해결하기 위해 토너먼트 방식의 선택 연산을 도입하였다. 이 방식은 구역을 나눠 그 구역에서 가장 좋은 2개의 유전자를 선택하여 다음 세대를 만드는 방식이다. 예를 들어, 100개의 유전자가 있다고 가정하고 이를 10개의 구역으로 나눈다면 한 구역 당 10개의 유전자들이 할당되고 10개의 유전자들 중 가장 좋은 2개의 유전자들이 선택된다. 이 선택 연산은 반드시 가장 좋은 유전자를 선택하지는 않는다. A구역에서 선택된 유전자가 B구역에서 선택되지 못한 유전자 보다 좋지 않은 점수를 기록했을 수 있다. 하지만 유전자가 다양하게 선택됨으로써 유전적 다양성이 확보된다. 따라서 세대가 지나감에 따라 유전자의 가중치들이 같아지는 현상을 감소시킬 수 있다.

또, 초기에 교차 연산에서 적합도를 내림차순으로 정렬시켜서 우수한 개체들 몇 개를 뽑아 다음 세대의 자식 개체들을 만들었는데, 이는 단순히 가중치의 위치만 뒤 섞어주는 연산으로 몇 세대만 지나도 지역 최대 문제에 빠져 가중치가 거의 같은 개체만 남아 버리는 문제가 있었다. 따라서 벡터 연산의 일종인 가중 평균

(Weight Average) 방식을 사용하여 선택된 개체 A와 개체 B의 적합도에 따라 편향을 갖는 형태로 자식 개체를 생성하도록 하였다. 이렇게 하면 교차 연산하려는 두 개체 중 더 큰 적합도를 갖는 방향으로 가중치를 취할 수 있다. 수식은 다음과 같다.

$$W' = WA \cdot Fitness(A) + WB \cdot Fitness(B)$$

적절한 가중치와 선택, 교차 연산이 설정되었다면 그 다음으로 중요한 것은 바로 가중치의 정확한 평가, 즉 정확한 적합도 평가이다. 초기 본 연구에선 유전자들이 한 번만 플레이 한 후 적합도 평가를 실시하였다. 그렇게 설정한 후 발전과정을 지켜보았을 때, 다음 세대로 넘어갔지만 플레이가 이전 세대보다 못하다 것을 발견하였다. 이유를 확인하기 위해 텍스트 파일에 기록된 적합도와 가중치들을 살펴보았을 때, 같은 가중치를 가지고 있지만 적합도는 더 낮게 기록된 부분을 발견하였다. 그 이유는 다음 블록이 무작위로 생성되기 때문에 우연히 좋은 플레이를 했을 가능성이 있기 때문이다. 그렇기에 이전 세대에서 좋은 가중치를 물려받은 유전자는 다음 세대로 넘어갈 때 선택되지 못 하고 다른 유전자가 선택되었고, 그렇게 선택된 유전자는 선택되지 못한 유전자보다 좋지 못할 때가 있었다. 그런 이유로 다음 세대로 넘어갔음에도 불구하고 더 좋지 못한 플레이를 보였다. 이는 블록이 무작위로 나오는 것이 플레이에 큰 영향을 미치기 때문이다. 같은 가중치 계수를 갖고 있어도 다음에 나올 블록이 무작위로 정해지기 때문에 잘 맞지 않는 블록들이 계속 나온다면 상황에 따라 기존의 플레이보다 못한 플레이를 할 수 있고 적합도 평가에서 낮은 점수를 받게 된다. 반대로 잘 맞는 블록들이 계속 나온다면 보다 오래 살아남아 높은 적합도 점수를 기록할 수 있다. 즉, 블록이 어떻게 나오느냐에 따라 유전자의 적합도가 다르게 평가받을 수 있다. 그래서 이것을 고려하지 않고 한 번의 플레이로만 적합도를 평가한다면 정확한 평가를 하지 못하게 되고 결국 올바른 방향으로 발전하기 어렵게 된다. 따라서 보다 정확한 적합도 평가를 위해 여러 번의 플레이, 한 유전자 개체에 적어도 100번의 플레이를 시킨 후에 그것의 평균을 평가하는 방식으로 변경하여 가능한 올바른 방향으로 발전할 수 있게 하였다. 변경 후 실행결과를 확인하였을 때, 발전된 세대가 퇴행하는 것 없이 최소한 이전 세대보다 계속해서 나은 플레

이 하는 것을 확인하였다.

성능을 평가할 때 시간복잡도라는 중요한 요소를 빼놓을 수 없다. 컴퓨터과학에서 알고리즘의 성능을 측정하기 위한 방법으로 시간복잡도를 사용한다. 시간복잡도는 입력 데이터 n에 의존하여 얼마나 반복되는지를 계산식에서 가장 영향이 큰 부분만 간단하게 표시해놓은 것이다. 본 논문에서도 알고리즘의 성능을 측정하기 위해 테트리스 AI의 플레이를 시간복잡도로 계산하였다. AI의 플레이에서 시간 성능의 핵심은 블록이 놓일 최적의 위치를 찾는 것이다.

<블록의 최적 위치를 찾는 과정>

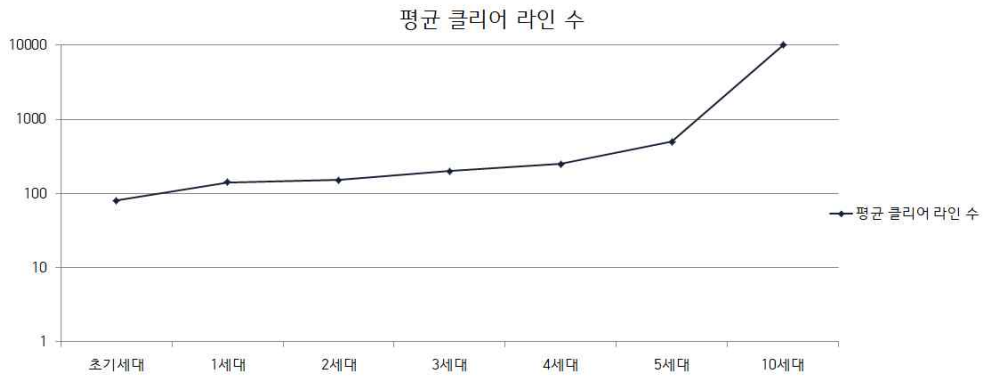
1. 현재 블록을 너비 10, 높이 20의 2차원 공간에서 x축으로 1씩 이동시킨다.
2. 블록이나 바닥이 닿을 때까지 그대로 y축으로 이동시킨다.
3. 각 위치에 대해 가중치를 계산한다.
4. x축을 가로 끝으로 이동했으면, 모양을 바꾼다.
5. 가장 적합도가 큰 곳을 찾기 위해 위의 과정을 반복한다.

알고리즘의 시간복잡도를 계산하기 위해선 크게는 블록을 각 위치에 놓아보는 연산과 블록을 놓았을 때 그 위치의 가중치를 계산하는 연산의 횟수를 세어보면 된다. 블록을 각 위치에 놓아보는 연산의 횟수를 계산해보면 맵을 이차원 배열로 표현했을 때 map[20][10]이고, 이중 for문을 이용하여 연산하기 때문에

```
for(행 크기만큼)
    for(열 크기만큼)
```

$O(n^2)$라고 생각할 수 있지만, 맵의 크기가 10x20으로 제한적이다. 블록은 가로로 1씩 이동하면서 놓이기 때문에 각 블록이 놓이는 횟수는 10번보다 많을 수 없다. 또 블록을 놓을 때, 회전한 블록이 기존의 블록이나 벽, 바닥에 닿는지를 확인하는 연산을 한다. 따라서 하나의 블록이 놓이는데 필요한 연산 횟수는 최악일 때 10(블록이 가로로 이동하는 위치) x 20(블록이 세로로 이동하는 위치) x 4(블록의 회전수)이므로 총 800회의 연산을 하게 된다. 또 가중치를 계산하는 연산은 블록이 놓일 때마다 맵의 크기만큼만 계산하기 때문에 최대 200회의 연산을 한

다. 최종적으로 800x200이므로 최악의 경우 160,000번에 가까운 연산을 한다. 현재 CPU로 위의 연산을 할 때, 위의 연산들은 시간을 지연시키는데 거의 영향을 미치지 않았다. 싱글 스레드 관점에서 실행시간은 0초에 가깝다. 즉, 시간복잡도는 맵의 크기에 따라서 달라진다고 볼 수 있지만, 테트리스 플레이의 규격을 준수하여 크기는 가로가 10이고, 세로가 20으로 고정되기 때문에 O(1)라고 봐도 무방하며, 입력 데이터에 상관없이 최대로 고정된 수치가 160,000을 가지므로 이를 최적화하여 수행횟수나 연산횟수를 줄이는데 참고할 수 있다.



[그림4-6] 세대에 따른 평균 클리어 라인 수

위의 그래프를 보면 일정 세대까지는 일정한 간격으로 증가하거나 때로는 더 좋은 개체를 만들지 못해서 머무르는 경향을 보인다. 그러나 대체로 10세대 근처에서 갑자기 효율이 급증하였다. 이때, 현실적인 속도로 보았을 때 약 일주일 정도를 끊임없이 플레이하므로 충분히 최적 해의 범주에 속한다고 볼 수 있다.

유명한 Opensource Tetris AI가 약 21만 줄까지 측정하다가 종료하였다고 한다. 사실상 임계점을 넘어가면, 끝나지 않고 계속해서 플레이한다. 따라서 10만을 넘긴 시점에서 최적화가 되었다고 판단하여 그 이상의 플레이를 종료하였다.

마지막으로 최적 세대라고 판단된 개체의 가중치들은 다음과 같다.

-0.088149 -0.164408 -0.845511 0.500310
최대높이 높이편차 막힌공간 완성된줄

가중치들의 비율을 보면, 마지막 가중치인 완전한 줄 수를 제외하면 - 값을 갖는 것을 알 수 있다. 음수를 갖는 가중치 중에서 블록에 막힌 공간의 수가 가장 낮았으며, 높이에 관련된 가중치들, 블록의 최대 높이와 높이 편차가 비슷하게 측정되었다. 이로써 가중치 간의 관계와 비율을 알 수 있다.

마지막으로 필자가 겪었던 시행착오에 대해 말하려 한다. AI를 완성하고 다른 테트리스 AI들과 비교했을 때, 큰 흐름에서 코드나 알고리즘 성능에서 차이는 없었지만, 게임을 계속해서 플레이하는 것에 있어서 큰 차이를 보였다. 필자가 개발한 AI는 오래 살아남기는 했지만 언젠간 블록이 쌓여 게임이 끝나 버리는 일이 계속해서 발생했다. 이 현상은 세대가 발전되어도 계속되었다. 그래서 오랜 시간 동안 다른 AI의 플레이를 관찰한 결과, 블록이 일정한 비율로 계속 나오는 것에 비해 필자가 개발한 AI는 블록이 정말로 무작위로 나온다는 사실을 발견했다. 그래서 테트리스 규칙을 찾아본 결과 크게 보았을 때, 테트로미노들이 일정하게 나오도록 하는 것도 게임규칙 중 하나라는 것을 알아내었다. 정확히 말하면, 이것을 Random Bag이라고 한다. 7개의 테트로미노들이 7번 안에 무조건 한 번씩 나오고, 차이는 단지 순서의 차이일 뿐이다. 이것을 코드에 추가하였더니, 게임이 끝나는 일 없이 계속해서 플레이하는 AI가 되었다. 이것을 계기로 필자는 영향을 줄 수 있는 규칙과 변수를 하나도 빠짐없이 고려하여 문제해결에 접근해야 올바른 답을 구할 수 있다는 것을 느꼈다.

5. 결론

본 논문에서는 유전 알고리즘을 이용하여 최적화된 테트리스 AI를 구현하였다. 이를 위해 적합한 요소들을 가중치로 선정하고 학습하는 과정을 보여주었다.

가중치의 개수에 따라서도 실행결과가 확연히 달라진다는 것을 알았다. 즉, 적합한 요소를 가중치로 설정하는 것이 중요하였다. 또 가중치들을 정확하게 연산하는 과정도 한 치의 오류가 없어야 했다. 블록의 모든 모양과 위치를 고려해야 하는데, 그렇지 않다면 가중치의 항목이 아무리 좋아도 결과로 나타나지 않았다. 선택과 교차, 변이 연산을 어떻게 설정할지 고민해야 한다. 본 논문에서는 토너먼트 기반으로 선택하였다. 초기에 교차 연산을 단순히 가중치들의 계수들을 바꿔주는 것에서, 벡터를 이용한 가중치 평균을 이용한 방법으로 연산하였다. 돌연변이 연산 시에는 가중치 전체를 완전하게 무작위로 정할지 아니면 특정한 가중치 하나만 무작위로 정할지 등에 대한 것들이 영향을 주었다. 마지막으로 유전 알고리즘이 게임을 플레이하는 것도 결국 사람이 플레이하는 것과 같은 과정이 들어가기 때문에 테트리스 게임 자체의 구현도 매끄러워야 했다. 특히 이 모든 과정에서 시간복잡도가 지나치게 우려되지 않도록 하였다. 또, 알려진 오픈소스 테트리스 AI와 비교하여 어느 정도 성능이 있는지 측정하였다.

향후 과제로는 최적 개체에 대한 조건을 더 엄격하게 설정한 후에 만족시키는 개체를 찾도록 하는 것과 최적 개체의 가중치 계수 값을 세밀하게 분석하여 상관관계와 우선순위를 정해야 한다. 유전 알고리즘이 복잡한 문제를 해결할 수 있음을 확인하였고, 널리 알려진 활용 방안들을 살피며 아직 특정하지 않은 문제에 적용할 여지가 많다고 본다.