러쉬 플레이 납품 자동 생성 - 세부 로직

문서 목적

- 러쉬 플레이 납품을 마을 레벨 고려하여 동적으로 생성하는 것

밸런스 목표

- 동일한 패턴이 계속 나오는 느낌을 주면 안됨
- 유저에게 불합리한 느낌을 주면 안됨
- 특정 생산 건물에 부하가 너무 적거나 많으면 안됨

기본 전략

- 일정 시간 동안 생산할 수 있는 생산품의 총량을 산출
- 해당 생산품들을 적절히 분배하여 다양한 납품 패턴 생성

용어 설명

용어	설명
기준 시간	• 위에서 언급한 일정 시간에 해당하는 단어 • 기본 값은 180초 (정확한 값은 RushDeliveryPolicy 시트의 Duration 컬럼 참조)
납품 그룹	• 기준 시간 동안 생성된 납품들의 집합 • 기준 시간 1개당 하나의 납품 그룹만 존재
총 생산량	• 기준 시간 동안 각 생산품 별로 생산된 총량
재료 차출량	• 기준 시간 동안 각 생산품이 재료로 사용된 양
최종 생산량	• 총 생산량 - 재료 차출량 • 기준 시간 동안 재료로 쓰이고 남은 양 의미
단독 부과	• 재료 생산 시간을 고려하지 않은 시간 부과량 ◦ 예시 : 쿠키의 단독 부과 시간은 30초

계층 부과	• 재료 생산 시간을 고려한 시간 부과량 ◦ 예시 : 쿠키의 계층 부과 시간은 30초(쿠키) + 40초(달걀) + 1.25초(밀) = 71.25초) **주의 사항** • 병렬의 시간 부과량은 CraftTime/슬롯 수임 (예시 : 달걀 120초 / 슬롯 6개 = 20초) • 작물의 경우 CraftTime / 밭의 수임 (예시 : 밀 15초 / 밭 24개 = 0.625초) • 단독 부과, 계층 부과 모두에 해당됨

전체 흐름 설명

1. 공장에서 다양하게 생산한다고 가정 및 유도
2. 일정 시간 동안의 총 생산량 산출
3. 납품을 몇 개 만들지 결정
4. 생산 물품을 작은 단위로 나눔 **(나눠진 단위를 세그먼트라고 명명)**
5. 나눠진 생산 물품을 각 납품에 분배

Step 별 요약

- Step 1: 유저의 마을 레벨 기반으로 러쉬 플레이에 참여할 생산 건물, 생산품 파악
- Step 2: CraftGroup을 참조하여 직렬 생산 건물의 계층 정리 (상위 공장 → 하위 공장 → 축사 → 밭)
- Step 3: 상위 공장부터 공장 내 생산 다양성 확보
- Step 4: 최종 생산량 산출 (납품에서 요구할 총 물량 산출)
- Step 5: 납품 개수, 납품 별 요구 물품 종류 수 결정
- Step 6: 최종 생산량을 세그먼트로 분리
- Step 7: 세그먼트를 각 납품에 분배

Step	설명	Pseudo code	C# 코드

Step 1

유저의 마을 레벨 기반으로 해금된 생산 건물, 생산품 파악

- 유저 레벨에 따라 러쉬 플레이에 참여하는 생산 건물과 생산품 리스트 파악 단계
  - Building, BuildingInstance, Item 시트 참조
  - Building, Item 시트에 **IsRushEligible** 컬럼 추가
    -> 러쉬 플레이 참여 유무 확인용

예외 처리: 밭 최대 개수

- 밭은 러쉬 플레이에 참여하지만, 일반 납품에도 참여하고 계속 개수가 늘어남
- 밭의 러쉬 플레이 참여 수를 제한하지 않으면 장기적으로 문제 발생할 수 있음
- 일단, 러쉬 플레이에 참여하는 최대 밭 수를 config로 지정함 (고도화 시 비율로, 또는 연산 결과로 결정)
  - Config: MAX_RUSH_FIELD

고도화 및 확장성 고려

고도화 (코드 설계 시 고려할 확장성)

- 현재는 러쉬 플레이 참여 유무에 대한 컬럼만 존재(IsRushEligible)
- 나중에 러쉬 플레이 생산 건물이나 생산품이 많아진다면 일부만 선별하는 작업 필요
  -> 너무 많은 생산 건물과 생산품이 참여하면 유저에게 혼란을 줄 수 있음)

코드

```
# Step 1
for each
    if (

# Step 1
for each
    if i

# Step 1
group by
    Crea

    }
```

출력 예시

```
List<Proc
예시:
{
  Type: '
  Categor
  Amount:
  Availab
}
```

코드

```
public
{
    pub
    pub
    pub
}

public
{
    pub
    pub
    pub
    pub
}

public
{
    pub
    pub
    pub
    pub
}

public
{
    pub
    pub
    pub
    pub
}

public
    int
    Lis
    Lis
    Lis
{
    //
    var
    for
    {

    }

    //
    var

    //
    var
```

<table>
<tr>
<td></td>
<td></td>
<td></td>
<td>
<pre>
var

//
var
for
{
}

ret
}
</pre>
출력 예시
<pre>
[
  {
    Bui
    Cat
    Bui
    Ite
  },
  {
    Bui
    Cat
    Bui
    Ite
  }
]
</pre>
</td>
</tr>
<tr>
<td>Step 2</td>
<td>
CraftGroup을 참조하여 직렬 생산 건물의 계층 정리

공장에서 뭘 생산할지 정하기 전에 어떤 공장이 상위 공장인지 파악 필요

계층 부과를 고려해서 뭘 생산할지 정하기 때문

공장에서 생산한 아이템이 다른 공장에서 재료로 쓰이는 경우가 있으면 하위 공장, 없으면 상위 공장으로 취급
최종 계층 구조는 상위 공장 → 하위 공장(사료 공장 제외) → 밭 제외 병렬(축사) → 사료 공장 → 밭

</td>
<td>
코드
<pre>
functio

use
for

res

for
</pre>
</td>
<td>
코드
<pre>
public
{
    Top
    Bot
    Par
    Fie
}

public
{
    pub
    pub
    pub
}
</pre>
</td>
</tr>
</table>

- 상위 공장, 하위 공장 간의 우선 순위는 UnLockLevel로 결정
- UnlockLevel이 높으면 우선 순위 더 높음
- UnlockLevel이 같으면 랜덤으로 처리

```
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31      retu
32
33
34
```

출력 예시

```
List of:
{
  Contex
  Tier: '
  Unlockl
}
```

```
15
16  public
17      Lis
18      Lis
19      Lis
20  {
21      //
22      var
23
24      );
25
26      var
27
28      for
29      {
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60      }
61
62      //
63      ret
64
65
66
67
68  }
```

출력 예시

```
1. Dairy
2. Baker
3. Cowsh
```

<table>
<tr>
<td></td>
<td></td>
<td></td>
<td>
<pre>
4. Chick
5. Field
</pre>
</td>
</tr>
<tr>
<td>Step 3</td>
<td>

상위 공장부터 공장 내 생산 다양성 확보 (최종 생산량 산출)

개념 및 필요성

- 납품 다양성은 공장 생산 다양성에 기반함
- 공장에서 다양하게 생산한다는 가정 하에 납품 생성
  - 유저도 기본적으로 다양하게 생성하는 경향 존재
    -> 다양하게 생산하지 않는 유저도 다양성을 강조하는 납품 패턴 때문에 다양하게 생산할 수 밖에 없음

기본 전제

- 상위 공장의 재료 차출량은 어떠한 상황에서도 충족된다고 가정
- 사료 공장처럼 연결된 상위 공장은 많은 경우에도 상위 공장의 수요를 충족시키지 못하는 경우는 없음

다양성 확보 대상

- 직렬 공장과 밭 뿐만 다양성 확보 대상임
  -> 축사는 1가지 생산품만 존재

다양성 확보 방법

- 기준 시간에서 재료 차출량 생산을 위한 시간을 빼고 남은 시간을 기준으로 다양성 확보
  - 예시 : 기준 시간 180초, 수프 공장이 크림 3개 차출(45초) → 유제품 공장의 남은 시간은 135초
  - 기준 시간은 RushDeliveryPolicy 시트의 Duration 컬럼 참조
- 남은 시간 내에서 각 생산 공장 별로 랜덤한 생산품 선정
  - 135초가 남은 상태에서 크림, 치즈 중 1개를 랜덤으로 선정

</td>
<td>
코드
<pre>
functior

    vali
    for

    if v

    else

functior
    for

    retu
</pre>
</td>
<td>
코드
<pre>
public
{
    Ite
    Cra
    Rem
    Rem
}

public
{
    pub
    pub
}

public
    Pro
    Dic
    Lis
    flo
{
    var

    for
    {
    }

    //
    ret
}
</pre>
</td>
</tr>
</table>

	◦ 이 과정을 계속 반복 ◦ 단, 남은 시간 보다 생산 시간이 큰 품목은 선택 불가 ▪ 예시: 27초 남았으면 크림만 선택 가능 ◦ 남은 시간 내에 어떠한 생산품도 생산할 수 없으면 반복 종료 • 자투리 시간 처리 방법 ◦ 생산 시간이 가장 짧은 생산품 선정(동률이면 랜덤으로 1개 선택) ◦ 남은 시간에 비례해 생산품을 확률적으로 추가 ▪ 예시: 3초 남았고 크림이 선정됨 → 크림의 생산 시간은 15초, 남은 시간은 3초 → 20% 확률로 크림 1개 추가 **우려 사항** • 확률로 처리하는 것은 완벽한 방안은 아님 (오류가 누적될 수 있음) • 러쉬 플레이가 연속된다면 남은 시간을 다음 기준 시간으로 넘겨주는 로직이 필요		
Step 4	**납품 개수, 납품 별 요구 품종 수 결정** • 현재는 RushDeliveryPolicy에 있는 고정 값 사용 ◦ VarietyCount1은 1종 요구 납품 수 의미 (3으로 고정) ◦ VarietyCount2은 2종 요구 납품 수 의미 (3으로 고정) ◦ VarietyCount3은 3종 요구 납품 수 의미 (3으로 고정) ◦ VarietyCount4~6은 0으로 고정 **고도화 : 고정 값이 아니라 비율 사용**	코드 1	코드 1
Step 5	**최종 생산량을 세그먼트로 분리** • 각 생산품의 최종 생산량을 기반으로 세그먼트 생성 ◦ 한 번에 많은 수량을 요구하는 단조로운 납품을 줄이고, **분산된 납품 구성**을 만들기 위한 핵심 단계	코드 1	코드 1

기본 전제 (아래 상황 발생하지 않도록 값 조정)

- 최종 생산량이 1이상인 생산품이 최소 3종 존재한다고 가정
- 모든 생산품의 최종 생산량 합계가 18 이상이라고 가정

Step 6-1: 필요한 세그먼트 수 결정

- Step 5의 납품 개수, 납품별 요구 품목 수에 종속됨
- 현재는 Step 5가 고정이므로 필요 세그먼트 수도 18개로 고정됨

Step 6-2: 분할 대상 세그먼트 선정

- 각 생산품의 최종 생산량의 단독 부과 시간 합계를 가중치로 사용하여 분할 대상 선정
- 예시
  - 우유 최종 생산량 6개, 쿠키 최종 생산량 4개, 당근 수프 최종 생산량 3개라고 가정
  - 단독 부과 시간은 60초, 120초, 180초
  - 그러면 가중치를 60, 120, 180으로 설정해서 분할 대상 선정
- 예외 처리
  - 최종 생산량이 2개 이상인 생산품만 선정 후보가 됨

Step 6-3: 선정된 생산품을 몇 개의 세그먼트로 분할할지 결정

- 최소 2개, 최대 4개의 세그먼트로 분리
  - 예외 처리: 당연하게도 최종 생산량보다 많은 수의 세그먼트로는 분리 불가
    (당근 수프 최종 생산량 3개인데 4개의 세그먼트로 분리 불가)
- 가중치를 통해 몇 개의 세그먼트로 분리할지 결정

	◦ 2분할, 3분할, 4분할 각각에 대한 가중치 config 참조 (SEGMENT_SPLIT_2~4) ◦ 예외 처리: 최종 생산량보다 큰 수의 세그먼트 가중치 무시 (당근 수프 최종 생산량 3개면 이분할, 삼분할 가중치만 가지고 계산) **Step 6-4: 각 세그먼트에 최종 생산량 분배** • 세부 1단계: 일단 모든 세그먼트에 1개씩 분배 (빈 세그먼트 발생하지 않도록) ◦ 예시: 우유 15개를 4개 세그먼트로 분리 → [1,1,1,1]로 분배하고 11개 남음 • 세부 2단계: 남은 최종 생산량을 세그먼트 수로 나눈 값 만큼 첫번째 세그먼트에 할당(소수점 버림 처리) ◦ 11개/4 = 2.75개 → 소수점 버려서 2개 ◦ [3,1,1,1]로 분배하고 9개 남음 • 세부 3단계: ◦ 2번째 세그먼트에서 남은 량을 랜덤으로 가져감 ◦ 0~9개를 균등한 확률로 선정 ◦ 5개를 선정했다고 가정하면 [3,6,1,1]이고 4개 남음 • 세부 4단계 ◦ 다음 세그먼트로 넘어가면서 3단계 반복 ▪ 남은 분배량이 0이 되면 중단 ◦ 마지막 세그먼트는 남은 분배량을 전부 가져감 ◦ 예시: [3,6,1,1]에서 세번째 세그먼트가 2개 선정 → [3,6,3,1]이 되고 남은 세그먼트가 남은 거 다 가져감 → [3,6,3,3]		
Step 6	**세그먼트를 각 납품에 분배** • 같은 품목이 서로 겹치지 않게 분배 ◦ 2종 요구 납품인데 우유가 분배된 상태에서 우유 또 분배하면 1종 납품이 됨 • 분배 로직 ◦ 무작위 세그먼트 선정해서 무작위 납품에 넣음	코드 1 출력 예시 1	코드 1 출력 예시 1

	◦ 단, 선택된 세그먼트에 이미 동일 품목이 있으면 다른 납품에 넣음 • 예외 처리 ◦ 특정 세그먼트가 아무 납품에도 들어가지 못하는 상황 발생 시 그냥 분배 처음부터 다시 시작 (정상 case 나올 때 까지 재시도 전략)		