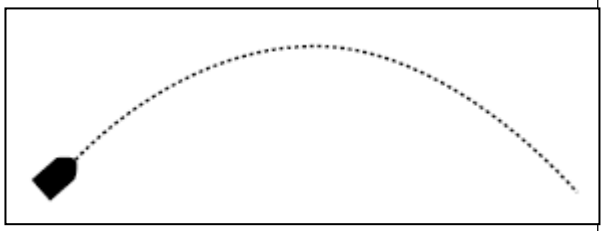
Physics Based Trajectory Prediction

- 3D movement
- Most common example:
  - Shooting or throwing objects
    - Characters should be able to shoot or throw accurately
    - Often, characters should be able to avoid incoming fire

Lecture Note: Game Algorithm 1 Movement 중

Physics Based Trajectory Prediction



- Projectile Trajectory
  - Absent air resistance, follows a parabola

$$\mathbf{p}(t) = \mathbf{p}(0) + s_m \mathbf{u} t + \mathbf{g}\frac{t^2}{2}$$

  - **p** – position at time *t*
  - **u** – firing position (normalized vector)
  - **g** – force of gravity
  - s_m – muzzle speed

the muzzle velocity (the speed at which the projectile left the weapon)

u is the direction the weapon was fired in
(a normalized 3D vector)

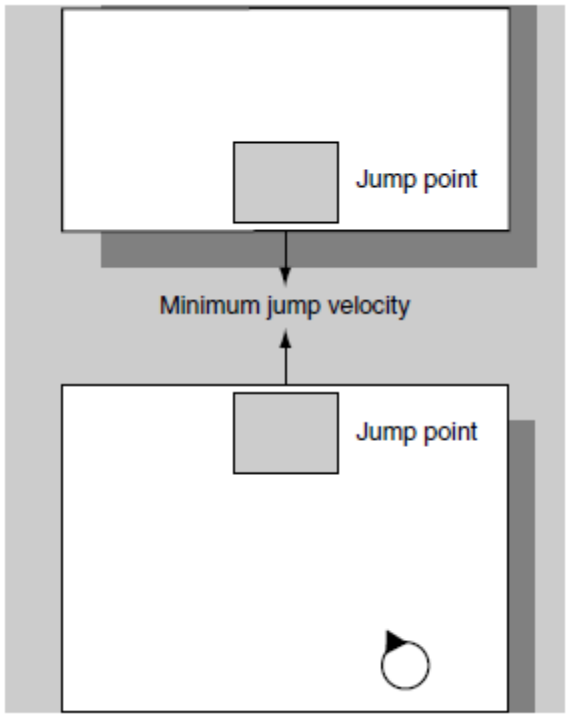
Physics Based Trajectory Prediction

- Predicting a landing spot
  - Calculate height (solve for Acomponent)
    - No solution: Projectile never reaches that height
    - One solution: Projectile reaches height at its vertex
    - Two solutions: Pick greater value
      - Greater value is for projectile descending.
      - If this value is negative, projectile has already passed the height and will not reach it again
  - Obtain A positions of this point
- Note: When a character wants to catch a ball, height should be at chest

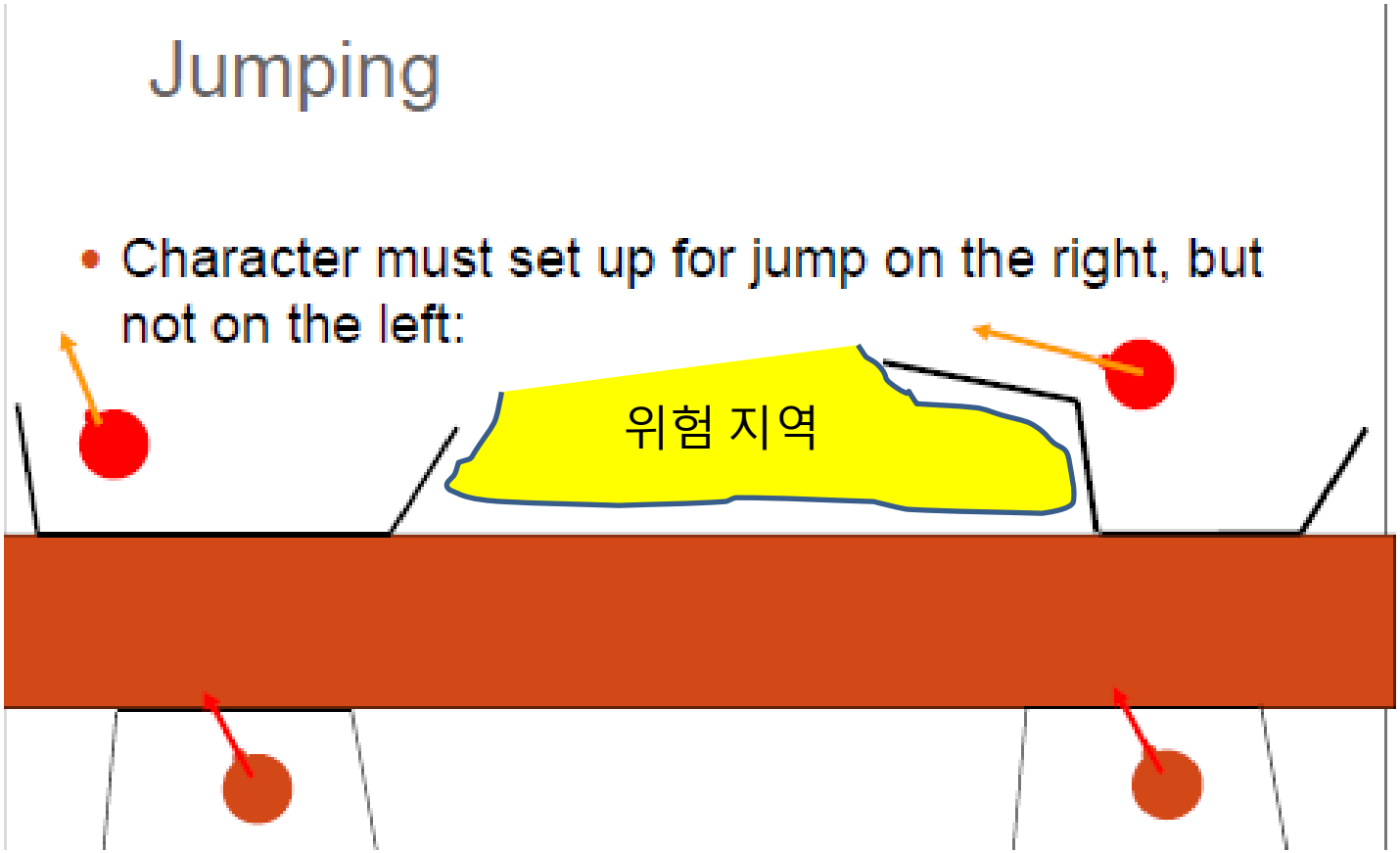
Py(t) = h 을 이용해서 풀이.
여기서 Py(t)는 P(t) 계산식에서 y성분만을 가지고 만든 계신식을 말함.

Jumping

- Shooter games need jumping
- Jumping is not part of regular steering mechanisms
- Jumps can be failures
  - Character tries to jump from one platform to another but fails
  - Cost of error larger (that causes the character to miss the landing spot and to plummet to the doom, for example) than for steering behavior.



proper velocity and direction for jumping



Landing area의 위치 및 모양을 고려한 jump action을 해야, 안전함

Jumping

- Character uses velocity matching steering mechanism to make the jump:
  - Character decides to take the jump
    - Pathfinding decides that character needs to jump,
      - this gives also type of jump

    OR
    - Simple steering mechanism drives character over edge,
      - this needs look ahead to determine type of jump
  - New steering behavior to do velocity matching to do jump
  - When character reaches jump point, new jump action is required

Jumping

- Jumps are difficult to make
  - if good landing area is small
- Create jump points for characters in level design
  - Player characters can try to make difficult jumps, but AI characters cannot
  - Minimize number of occasions where this limitation becomes obvious
    - Level design needs to hide weaknesses in the AI

Jumping

- Alternative uses pairs of jump points and landing pads
  - When character decides to make the jump, add additional step:
    - Use trajectory prediction code to calculate velocity required to reach landing area
      - This allows characters to take their own physics (weight, load, strength) into account
    - Use velocity matching steering algorithm

P(t) = jump후도착지점좌표 ➔ jumping velocity

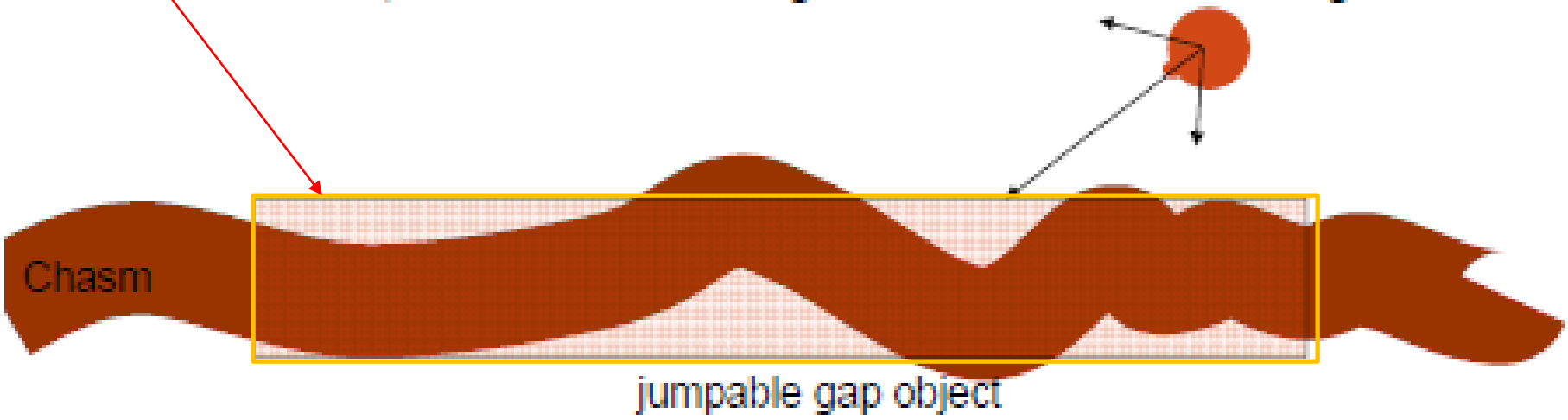
Jump Links

- Many developers incorporate jumping into their **pathfinding framework**.

- As part of the pathfinding system, we create a network of locations in the game. A connection between two nodes on either side of a gap is labeled as requiring a jump

Jumping

- Hole Fillers
  - Create an invisible *jumpable gap* as part of certain obstacles
  - Change obstacle avoidance behavior:
    - When detecting collision with a jumpable gap, character runs towards gap at full speed.
    - Just before the gap, character leaps into air
- Works well if landing area is large
- Fails for small landing areas
  - In this case, ensure that level design does not have small landing areas.



Coordinated Movement

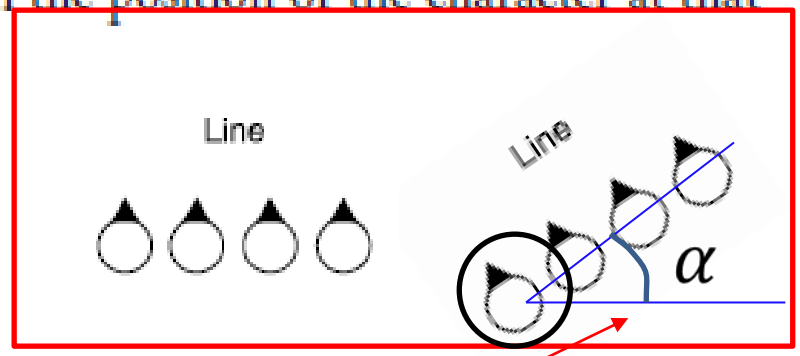
- Done by *groups* of characters
- Coordinated movement
  - can result from individual decisions
  - can result from decisions made by group as a whole

Coordinated Movement

- **Fixed formations** (고정된 대형)
  - Usually with a designated leader
  - Leader moves as an individual, everybody else moves based on leaders position
  - Actual position depends on number of characters

If a slot is located at r_s relative to the leader's slot, then the position of the character at that slot will be

$$p_s = p_l + \Omega_l r_s,$$

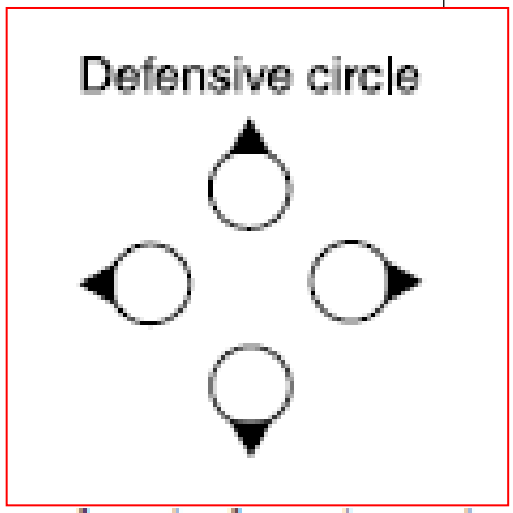


where p_s is the final position of slot s in the game, p_l is the position of the leader character, and Ω_l is the orientation of the leader character, in matrix form.

위의 예에서는, "시계반대방향으로 α 만큼의 회전 나타내는" 회전 매트릭스

The orientation of the character in the slot will be

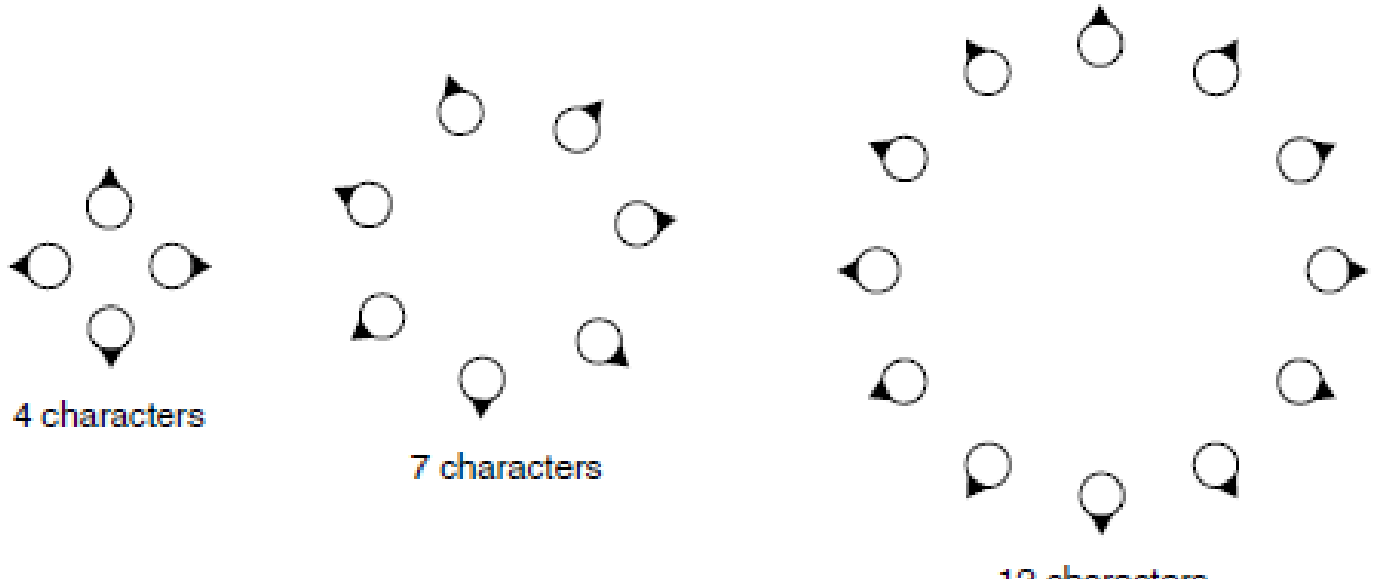
$$\omega_s = \omega_l + \omega_s,$$



where ω_s is the orientation of slot s, relative to the leader's orientation, and ω_l is the orientation of the leader.

Scalable Formations

A function can dynamically return the slot locations, given the total number of characters in the formation, for example.

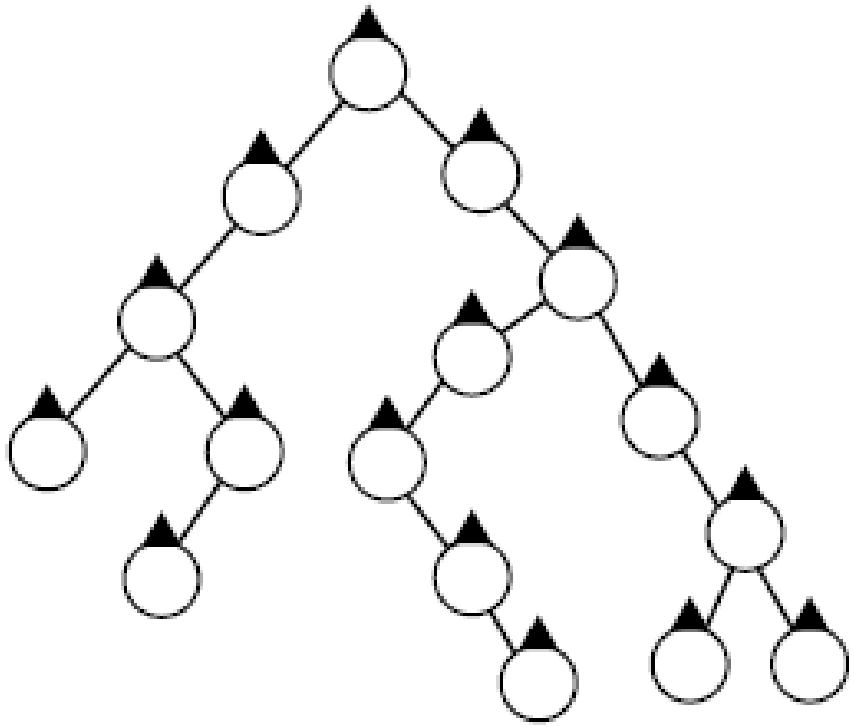


A defensive circle formation with different numbers of characters

Coordinated Movement

- Emergent Formations
  - Each character has its own steering system
    - Arrive Behavior
  - Characters select their target based on other characters in group
    - Allows characters to avoid obstacles individually
    - Difficult to get rules that do not lead to pathological formations

There is no overall formation geometry in this approach, and the group does not necessarily have a leader (although it helps if one member of the group isn't trying to position itself relative to any other member). **The formation emerges from the individual rules of each character**.



Emergent arrowhead (V-shaped) formation

(1)We can force each character to choose another target character in front of it and select a steering target behind and to the side, for example.

(2) If there is another character already selecting that target, then it selects another.

(3) Similarly, if there is another character already targeting a location very near, it will continue looking.

(4) Once a target is selected, it will be used for all subsequent frames, updated based on the position and orientation of the target character.

(5) If the target becomes impossible to achieve (it passes into a wall, for example), then a new target will be selected.

Dynamic Slots and Plays

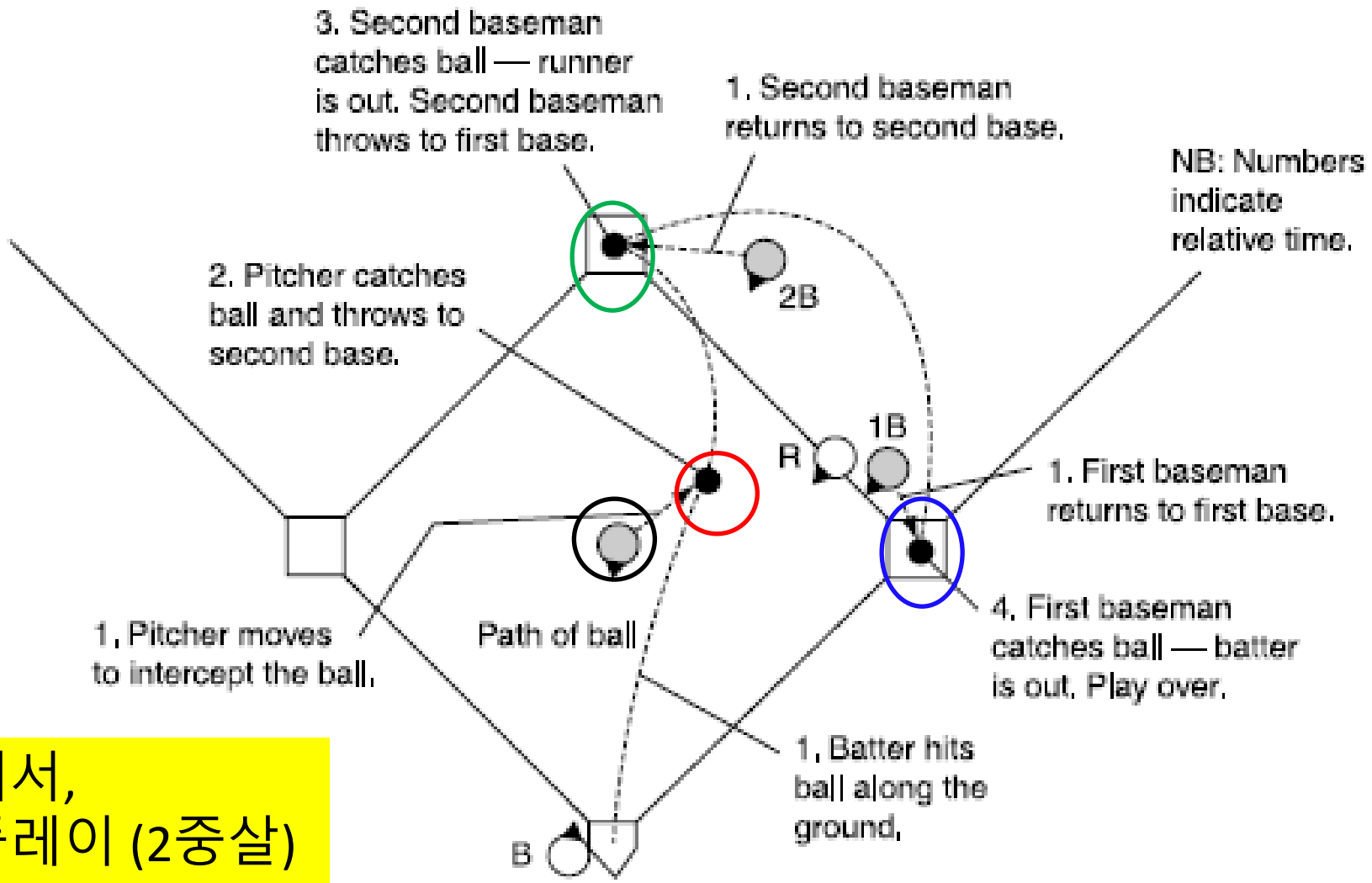
Generally a formation is a fixed 2D pattern that can move around the game level.

Slots in a pattern can be dynamic, moving relative to the anchor point of the formation.

This is useful for introducing a degree of movement when the formation itself isn't moving, for

implementing set plays in some **sports games**, and for using as the basis of **tactical movement**

Dynamic Slots and Plays



야구에서,
더블플레이 (2중살) 예제

A baseball double play

- Initially, they are in a fixed pattern formation and are in their normal fielding positions
- When the AI detects that the double play is on, it sets the formation pattern to a dynamic double play pattern.
- The slots move along the paths shown, bringing the fielders in place to throw out both batters.

음성 설명 없음.

앞 페이지에서의 음성 설명을 참조

음성 설명 종료