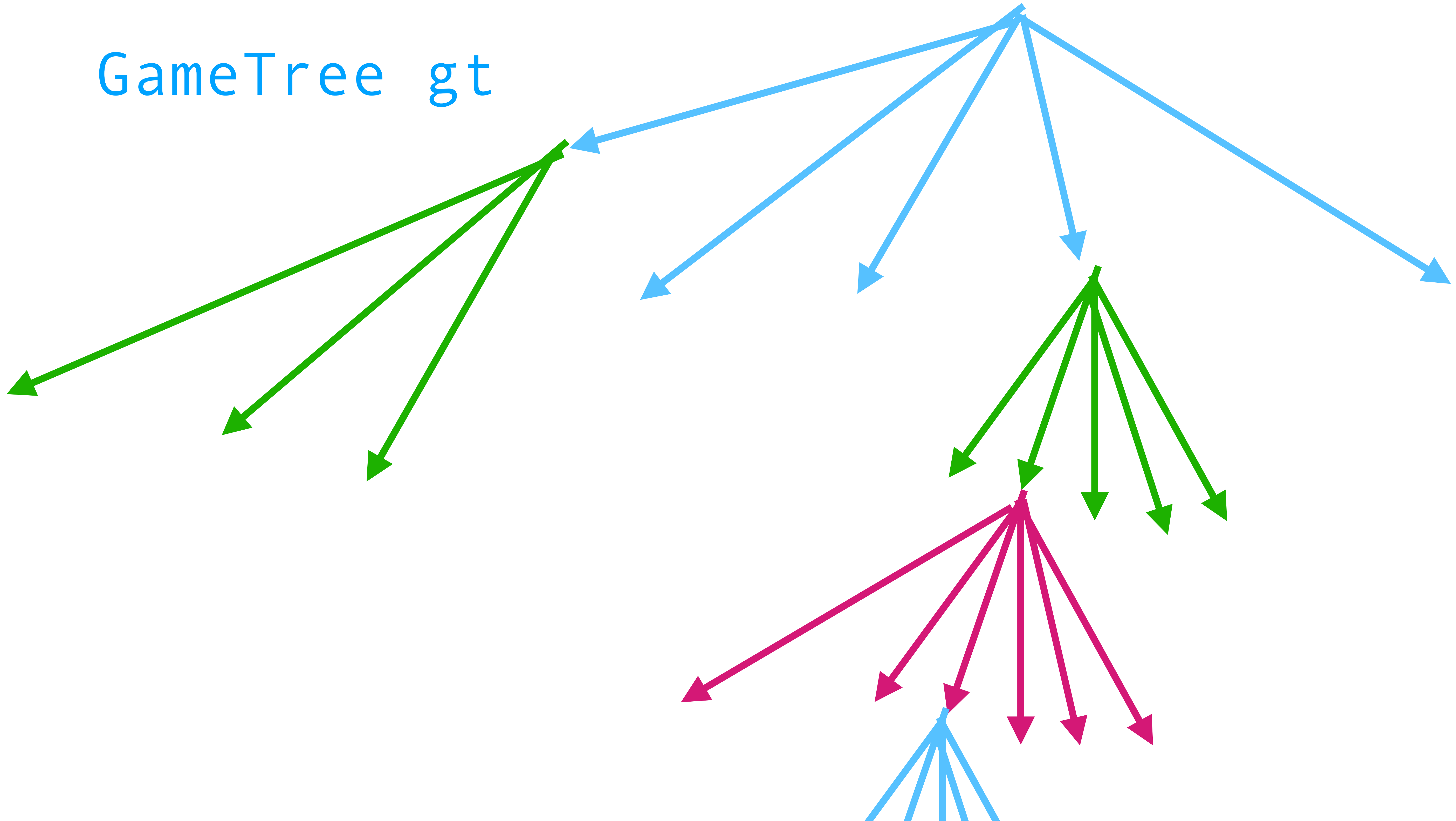
How to Use Game Trees

```
class Player

  GameTree gt;

  IAction takeTurn(GameState currentState, IAction actions[])
    gt = new GameTree(currenState)
    return this.strategy(gt)
```

Let’s go graphical

```
class Player

  GameTree gt;

  IAction takeTurn(GameState currentState, IAction actions[])
    gt = new GameTree(currenState)
    return this.strategy(gt)
```

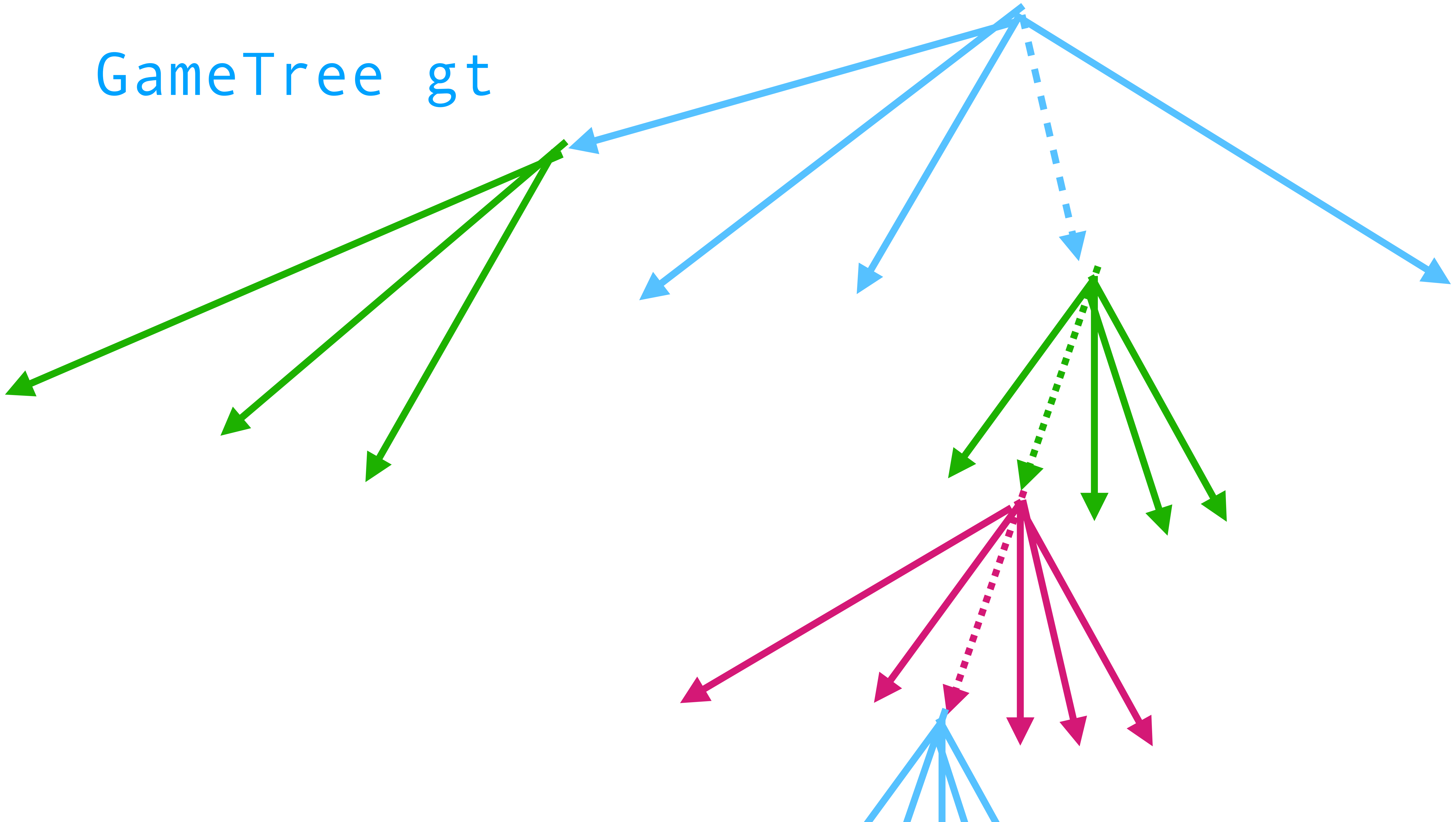
Assume 3 players:



GameTree gt

```
class Player

  GameTree gt;

  IAction takeTurn(GameState currentState, IAction actions[])
    gt = new GameTree(currenState)
    return this.strategy(gt)
```

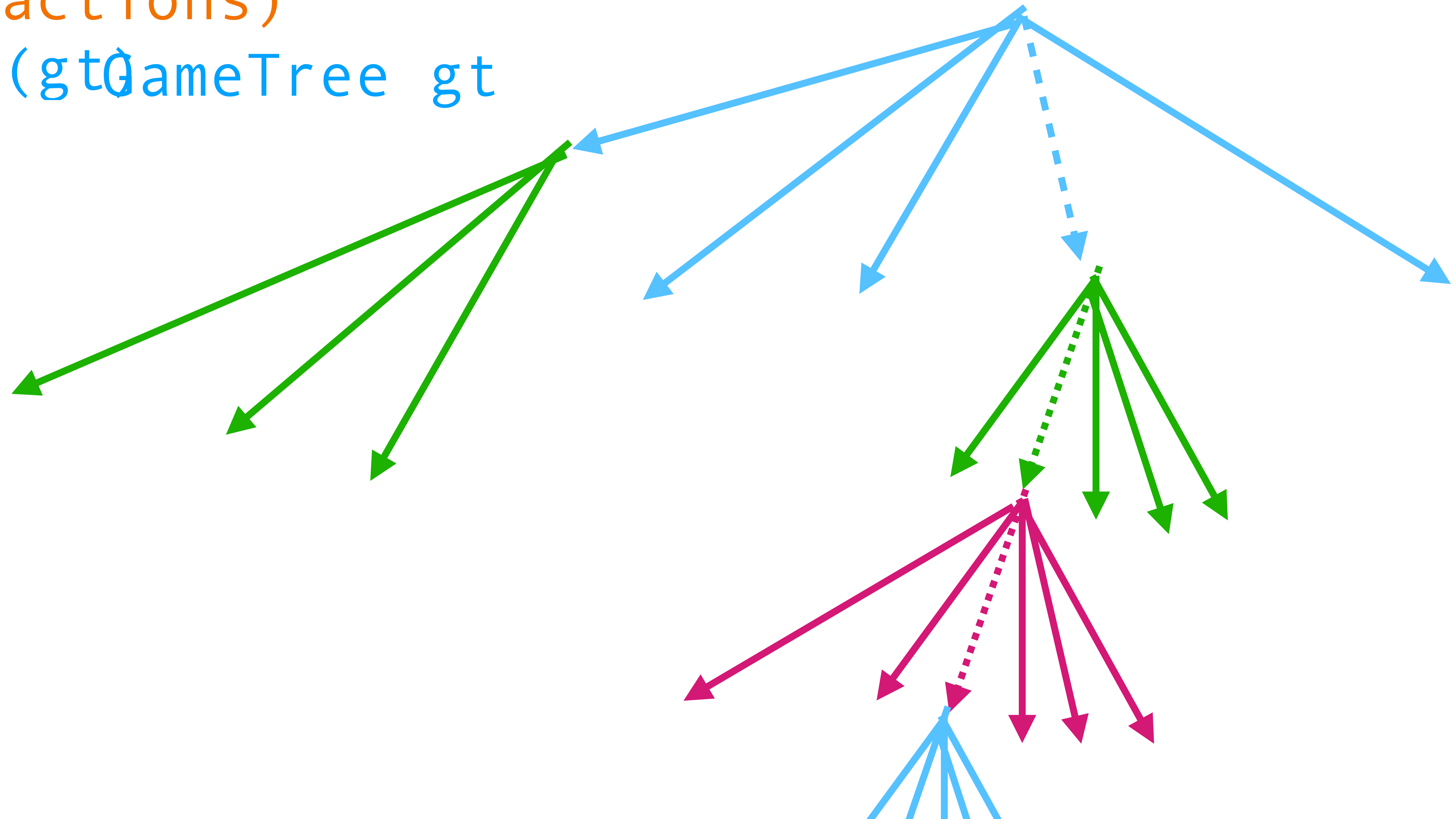
Assume 3 players, so each round has 3 actions:



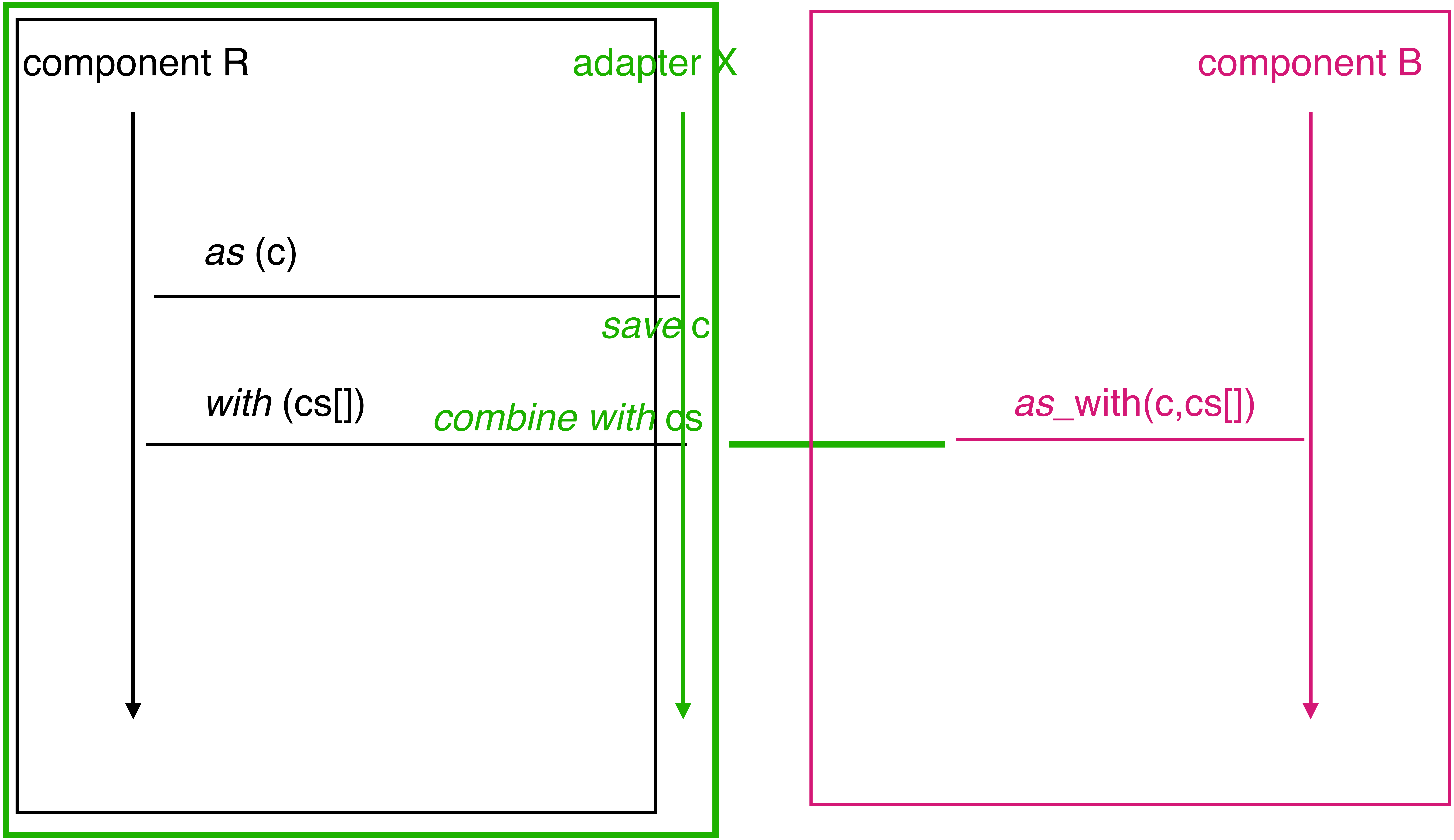
GameTree gt

```
class Player

  GameTree gt;

  IAction takeTurn(GameState currentState, IAction actions[])
    if (actions.size = 0)
       gt = new GameTree(currenState)
    else
       gt = gt.walkTree(actions)
    return this.strategy(gt)GameTree gt
```

Assume 3 players,
so each round has
3 actions:



How to Adapt Players


component R
adapter X
component B
as (c)
save c
with (cs[])
combine with cs
*as*_with(c,cs[])

Let’s go textual

Server

```
accept tcp connection into (in, out)
rpp = new RemoteProxyPlayer(in,out)
adp = new AdapterPlayer(rpp)
ref.register(adp)
```

Adapter

```
RemoteProxyPlayer r

IAction placePenguin(GameState gs) { … }

IAction takeTurn(GameState gs) { … }
```

Referee

```
GameState g;

Result runGame()
  ..
  nextAction = player.takeTurn(state)
  g = g.apply(nextAction)
  ..
```

Adapter

```
RemoteProxyPlayer r

GameState previous
Queue<Action> actions = new Queue(player#)
IAction takeTurn(GameState current)
   if (previous.player#() > current.player#)
      actions = new Queue(current.player#)
   else
      IAction step = current.difference(previous)
      actions.enqDeq(step)
   return r.takeTurn(current, actions.toActionsList())
```

Adapter

```
#; {GameState GameState -> Action}
;; determine which action takes the old state to the new one, if any
(define (diff-state state-old state-new)
  (define players-old (fishes-players state-old))
  (define players-new (fishes-players state-new))

  (define places-old (apply set (iplayer-places players-old)))
  (define places-new (apply set (iplayer-places players-new)))

  (list (set-first (set-subtract places-old places-new))
        (set-first (set-subtract places-new places-old))))
```