Game Balancing - Relay Race example

"Warm Fuzzy makes Salad"

Dale Wick
AdamCon 17
July 17, 2005

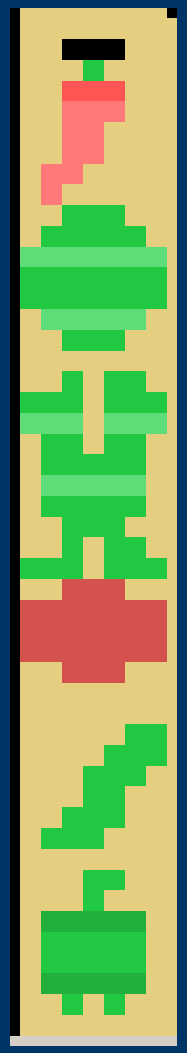
Relay Race Game Rules

- Relay race rules
  - Each participant picks up a goal combination of vegetables, and put them in that player's basket
  - The participant can carry up to two items at a time
  - For each level there are specific vegetables that are the goal for the salad.
  - Once all ingredients are captured by one player, the level is scored.
  - (Some vegetables are worth more than others, or are more rare than others, non-ingredients are worthless.)

Appearance

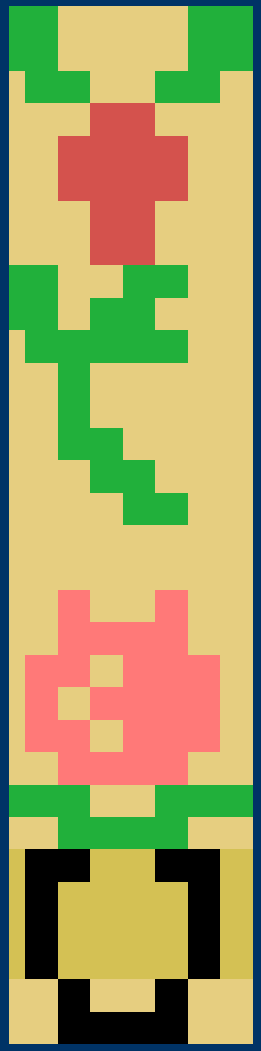
- Map appearance is a garden with walls, water, bushes and vegetables
- The stats are at the top show inventory
  - Holding (up to 2), Has (up to 4) and needing (up to 4)
- The target baskets are black squares

Some Vegetables



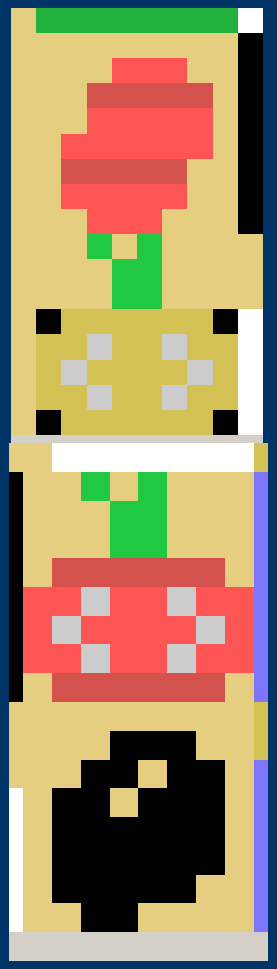
- Carrot
- Lettuce
- Celery
- Tomato
- Cucumber
- Green Pepper

More Vegetables and Fruit



- Radish
- Parsley
- Orange
- Pineapple

Enough Vegetables and Fruit



- Potato
- Pineapple
- Apple
- Black olive

The Actors Appearance

- Each contestant is represented by a sprite which is animated
  - up, down, left, right
  - Movement is in 8 directions
- Fire buttons control “has”
  - Left fire button picks up/drops in left paw.
  - Right fire button picks up/drops in right paw.

Possible Refinements and Restrictions

- Could add many levels, we will work with just 1.
- Could add non-passable barriers.
- There is always 3 of any needed ingredient to prevent hoarding, and only one of each is needed.
  - If an ingredient that is not needed can't be dropped on the players home square
  - If the player delivers one to that player's basket, then grabs the other two, the play is deadlocked with neither player able to complete the level until one of the remaining two ingredients is dropped somewhere on the map.

General Strategy

- P vs. P is pretty straight forward symmetric relationship
  - This competition is automatically balanced and playable against players of similar skill if garden is prepared fairly.
  - Contains true head to head competition, since players can rearrange locations of ingredients, or some can be closer to the player than others
- P vs. C requires some pacing and some artificial smarts
  - Need shortest path to get ingredient
  - Need heuristic to determine which ingredient to get, and when to drop in basket

Non-player Character (NPC) Overview

- Pseudo code for overall strategy computer strategy, to complete level fastest:
  - If holding two items, return them to basket
  - If at basket and holding items, drop items in basket
  - Locate closest needed ingredient (in “need” list, not “has” or “hold” lists). If two needed ingredients are the same distance away, choose one at random.
  - Fixate on traveling to chosen ingredient unless ingredient is captured by opponent or actor is next to it.
  - If next to it, pick it up in a free hand.

NPC Enhancements

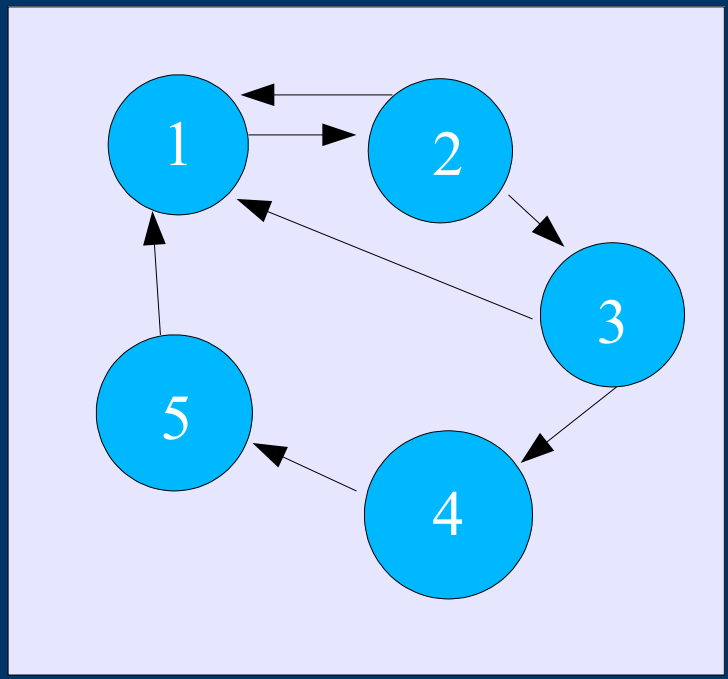
- Strategies the computer might choose:
  - Re-arrange one key ingredient to make it tougher to complete
  - Pace choices based on level
  - Weighted based on the level the computer might choose a wrong ingredient, or might pick up two of the same ingredient, thus wasting trips

Checkpoint to explore.

- How could the game be modified to employ vast resources?
- How could the game be modified to demonstrate asymmetric relationship using plastic asymmetry?
- How could the game be modified to provide triangularity?

Finite State Machine

- Keep track of the mode of thinking by a “state”
- Each time the character updates, perform an action based on the state: move, pick up, drop off
- The states are:
  - 1 – seek ingredient
  - 2 – travel to ingredient
  - 3 – standing at ingredient
  - 4 – travel to basket
  - 5 – standing at basket



Data structure for FSM

```
struct Actor {
    char id,frame,dir;      /* sprite id, visible frame, direction */
    char x,y;     /* Position on map */
    char hold[2];   /* What the actor is holding in its paws */
    char has[4];    /* What the actor has in its basket */
    char need[4];   /* The closeness to the win condition */
    long score;
    char isNPC;     /* Is played by real player or not */
    char state;     /* for NPC, what it thinking */
    char targetx,targety;   /* Where the NPC is walking to */
    char target;    /* The ingredient that the actor is after */
};
```

State 2 – Travel to ingredient or state 4 – Travel to basket

- Case 2 and 4:

```
void walk_towards(Actor *actor) {
    if(actor->target>0) {
        if(get_char(actor->targetx/8,actor->targety/8)!=actor->target) {
            actor->state=1;      /* The thing we're targeting is gone, sostate 1 */
            return;
        }
    }
    if(actor->targetx==actor->x && actor->targety==actor->y) {
        actor->state=actor->state==2?3:5;
    }
    actor->dir=choose_direction(actor->x,actor->y,actor->targetx,actor->targety);
}
```

Choose a direction to walk

```
char choose_direction(char fromx,char fromy,char
   tox,char toy) {
     if(fromx==tox && fromy==toy) return DIR_NONE;
     if(fromx==tox) return fromy<toy?DIR_DOWN:DIR_UP;
     if(fromy==toy) return fromx<tox?DIR_LEFT:DIR_RIGHT;
     /* Now do diagonal choices */
     if(fromx<tox) {
          return fromy<toy?DIR_UPLEFT:DIR_DOWNLEFT;
     }
     /* fromx>tox */
     return fromy<toy?DIR_DOWNLEFT:DIR_DOWNRIGHT;
}
```

State 3 – standing at ingredient

```
void standing_at_ingredient(Actor *actor) {
    char c=get_char(actor->x/8,actor->y/8);
    if(c!=actor->target) {
        actor->state=1; /* Wild goose chase, so make a new plan */
        return;
    }
    if(hold[0]==0) hold[0]=c; else if(hold[1]==0) hold[1]=c;
    put_char(actor->x/8,actor->y/8,0x8c)
    if(hold[0]!=0 && hold[1]!=0)
        actor->state=4;
    else
        actor->state=1;
}
```

State 5 – standing at basket

```
void standing_at_basket(Actor *actor) {
    if(hold[0]!=0) {
        drop_left(actor);
    }
    if(hold[1]!=0) {
        drop_right(actor);
    }
    actor->state=1;
}
```

Drop left

```
void drop_left(actor) {
	if(hold[0]==0) return;
	if(on_basket(actor)) {
		int i;
		for(i=0;i<need[i];i++) {
			if(need[i]==0) {
				need[i]=hold[0];
				hold[0]=0;
				return;
			}
		}
	} else {
		put_char(actor->x/8,actor->y/8,hold[0]);
		hold[0]=0;
	}
}
```

On Basket

```
int on_basket(Actor *actor) {
    if(actor->y<10) return 0;
    if(actor->y>13) return 0;
    if(actor->id==1) {
        if(actor->x<6) return 1;
    } else {
        if(actor->x>28) return 1;
    }
    return 0;
}
```

State 1 seek

- Search map for ingredients
- Keep track of near by ones
- Choose nearest one somehow (with randomness)

Code for State 1

- Pseudo code:
  - Make list of interesting needed ingredients
  - Loop through squares on the map
    - Check for a matching ingredient
  - Keep closest needed ingredient
  - Return target in actor structure

Implementation of state 1

```
void seek_ingredient(Actor *actor) {
    char i,x,y,count;  /* counters, the number of ingredients */
    char want[4];
    char closest=0,closestx=0,closesty=0,closestdist=127;

    for(i=0,count=0;i<4;i++) {
        char n=actor->need[i];
        if(n!=0 && n!=actor->has[0] && n!=actor->has[1]) {
            want[count++]=n;
        }
    }

    ... (continued next slide) ...
```

Implementation of state 1 (continued)

```
...
for(y=6;y<24;y++) {
      for(x=2;x<32;x++) {
            char c=get_char(x,y);
            for(i=0;i<count;i++) {
                  if(c==want[i]) {
                        char dist=calcdistance(actor->x,actor->y,x,y);
                        if(dist<=closestdist) {
                              closestx=x; closesty=y; closest=c; closestdist=
                        }
                  }
            }
      }
}
...(continued on next slide)...
```

Implementation of state 1 (continued)

```
    ...
    if(closest==0) {
        return;    /* Stay in state 1 */
    }
    actor->target=closest;
    actor->targetx=closestx;
    actor->targety=closesty;
}
```