PROJECT

Redux Road

We've learned the core concepts of Redux. Now it's game time.

In this project you will build an adventure game using reducers, state, and actions. The state will represent, well, the state of the game. It contains the player's inventory, distance travelled, and time on the road. Each event in the game is represented as an action. Players can gather supplies, travel, and–if they play risky–sometimes tip over the wagon carrying their supplies.

Let's get started!

Tasks

13/13 Complete

Mark the tasks as complete by checking them off

Initial State and Reducer

1.

First, define the starting point of our game. The player begins on day 0 at kilometer 0 with 100 units of supplies.

Define an `initialWagonState` with three properties:

- `supplies` starting at 100
- `distance` travelled, starting at 0
- `days` on the road, starting at 0

Hint

Your initial state should be a plain JavaScript object. For example:

```
const playlistState = {
  songs: ['claire de lune', 'take five', 'halo'],
  playMode: "shuffle"
};
```

2.

Define an empty reducer that will manage the state of the game. You can give it any name you prefer.

Like any Redux reducer, it should be a function with `state` and `action` parameters. It should set `state` to `initialWagonState` if no value is provided.

Hint

Here's an example function with a default parameter set to 0:

```
const someFunction = (value = 0) => {
};
```

3.

Add a `switch` statement to your reducer. The `default` case should return the `state`.

Hint

Use a `switch` statement with a `default` case. It should switch on the `action.type` value:

```
switch (action.type) {
  default: {
    return state;
  }
}
```

4.

A player may gather supplies, which takes them a day and doesn't cover any distance.

If the `action.type` is `'gather'`, return a new state object with:

- 15 more supplies
- The same distance
- 1 more day

Make sure to return a new object to comply with the three rules of reducers.

Hint

Add another case to your `switch` statement, making sure to use the spread operator (`...`). This example has `'eat'` and `default` cases:

```
switch (action.type) {
  case 'eat': {
    return {
      ..state,
      food: state.food - 10
```

```
    };
  }
  default: {
    return state;
  }
}
```

5.

A player may travel for any number of days, which costs 20 supplies for each day but adds 10 kilometers each day.

If the `action.type` is `'travel'`, assume that the `action.payload` contains the number of days to travel. Return a new state object with:

- 20 less `supplies` for each day
- 10 more kilometers of `distance` traveled for each day
- The number of days added to `days`

Hint

Use multiplication to calculate the new values. For example, if the player travels 2 days:

```
supplies: state.supplies - (20 * 2)
```

6.

If a player drives off-road or across deep rivers, the wagon may tip! You'll need to spend some supplies and a day to fix it.

If the `action.type` is `'tippedWagon'`, return a new state object with:

- 30 less `supplies`
- The same distance
- 1 more day

Hint

Add another case to your `switch` statement, making sure to use the spread operator (`...`). This example has `'eat'` and `default` cases:

```
switch (action.type) {
  case 'eat': {
    return {
      ..state,
      food: state.food - 10
```

```
    };
  }
  default: {
    return state;
  }
}
```

Play!

7.

Let's try our game out.

Start a game by calling the reducer with an `undefined` state and empty action object and storing the result in a new variable called `wagon` (Initialize it with `let`). Then print the `wagon` value to the console.

Our initial `wagon` state should look like this:

```
{
  supplies: 100,
  distance: 0,
  days: 0
}
```

Hint

Call the reducer with `undefined` and `{}` as arguments to get the initial state of the wagon.

8.

Our first day will be dedicated to travel.

- Call the reducer with the `wagon` state and an action with `type: 'travel'` and `payload: 1`.
- Store the returned new state back into `wagon`.
- Print the new state.

Our `wagon` state should look like this:

```
{
  supplies: 80,
  distance: 10,
  days: 1
}
```

Hint

You can overwrite your state variable like so:

```
let myState = reducer(undefined, {});
myState = reducer(myState, action);
```

9.

On the second day, we stop to gather supplies.

- Call the reducer with the new `wagon` state and an action with `type: 'gather'` and no payload.
- Store the new state back into `wagon`.
- Print the new state.

Our `wagon` state should look like this:

```
{
  supplies: 95,
  distance: 10,
  days: 2
}
```

Hint

You can overwrite your state variable like so:

```
let myState = reducer(undefined, {});
myState = reducer(myState, action1);
myState = reducer(myState, action2);
```

10.

On the third day, we try to ford a rushing river...and our wagon tips over, spilling some supplies.

- Call the reducer with the new `wagon` state and an action with `type: 'tippedWagon'`.
- Store the new state back into `wagon`
- Print the new state.

Our `wagon` state should look like this:

```
{
  supplies: 65,
  distance: 10,
  days: 3
}
```

Hint

You can overwrite your state variable like so:

```
let myState = reducer(undefined, {});
myState = reducer(myState, action1);
myState = reducer(myState, action2);
```

11.

On the following day, we set out for a long 3 days of travel.

- Call the reducer with the new `wagon` state and an action with `type: 'travel'` and a `payload: 3`.
- Store the new state back into `wagon`.
- Print the new state.

Our final `wagon` state should look like this:

```
{
  supplies: 5,
  distance: 40,
  days: 6
}
```

Additions and Extra Credit

12.

Currently, the player can continue traveling even if their supplies are negative! Fix this in the `'travel'` case.

If there are not sufficient supplies to travel the given number of days, return the current state.

Hint

Within the `'travel'` case, calculate the new supplies using the `action.payload`. If supplies are negative, return the current state. Otherwise return the new state with the supplies, distance, and days updates.

13.

Well done! You've taken great strides on the Redux Road. Watch your supplies and watch out for snakes!

If you'd like to keep practicing, try implementing these features:

- Add a `cash` property, beginning with `200` for the initial state
- Add a `'sell'` case: Players can give away 20 supplies to gain 5 cash
- Add a `'buy'` case: Players can gain 25 supplies at the cost of 15 cash
- Add a `'theft'` case: Outlaws steal half of the player's cash

reduxRoad.js

```
const initialWagonState = {
  supplies: 100,
  distance: 0,
  days: 0
}

const wagonReducer = (state = initialWagonState, action) => {
  switch (action.type) {
    case 'gather': {
      return {...state, supplies: state.supplies + 15, days: state.days + 1
      }
    }
    case 'travel': {
      return {...state, supplies: state.supplies - 20*action.payload, distance: s
tate.distance + 10*action.payload, days: state.days + action.payload}
    }
    case 'tippedWagon': {
      return {...state, supplies: state.supplies - 30, days: state.days + 1}
    }
    default:
      return state;
  }

}

let wagon = wagonReducer(undefined, {});
console.log(wagon);

wagon = wagonReducer(wagon, {type: 'travel', payload: 1})
console.log(wagon);

wagon = wagonReducer(wagon, {type: 'gather'})
console.log(wagon);

wagon = wagonReducer(wagon, {type: 'tippedWagon'})
console.log(wagon);

wagon = wagonReducer(wagon, {type: 'travel', payload: 3})
console.log(wagon);
```

```
{ supplies: 100, distance: 0, days: 0 }
{ supplies: 80, distance: 10, days: 1 }
{ supplies: 95, distance: 10, days: 2 }
{ supplies: 65, distance: 10, days: 3 }
{ supplies: 5, distance: 40, days: 6 }
```