One possible solution is to use value iteration, starting from Figure 17.4 from the textbook. We first modify it to use rewards that are a function of the chosen action as well, and we set $\epsilon = 0$:

1: **function** VALUE-ITERATION(mdp) **returns** a utility function
   inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s'|s,a)$, rewards $R(s,a)$, discount γ
   local variables: U, U', vectors of utilities for states in S, initially zero
   U_A, vector of utilities over states in S and actions in $A(\cdot)$, initially zero
   δ, the maximum change in the utility of any state in an iteration
2: **repeat**
3: $U \leftarrow U'$; $\delta \leftarrow 0$
4: **for all** states $s \in S$ **do**
5: **for all** actions $a \in A(s)$ **do**
6: $U_A[s,a] \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a)U[s']$
7: $U'[s] = \max_{a \in A(s)} U_A[s,a]$
8: **if** $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
9: **until** $\delta = 0$
10: **return** U

The loop through all possible states (Line 4) is an obvious obstacle to scalability. One question to ask is what are the conditions under which we can skip some states. The answer is that we can skip any state for which Line 6 would cause no change to $U[s,a]$. Looking at Line 6 more closely, we observe that $R(s)$, $A(s)$, and $P(s'|s,a)$ do not change, so the only possible change is in $U[s']$. In other words, on each iteration, we need to consider only the states s from which the agent can immediately move to a state whose U value changed in the previous iteration. For arbitrary MDPs, it is possible that all of the states' U values change on every iteration, which is why the textbook pseudocode loops through every state.

However, in our GridWorld MDPs, there is more limited connectivity among our states. We introduce a set C to keep track of which states' U values change on each iteration. We can also use the specific reward structure for a smarter initialization of our utility tables. To take advantage of these domain-specific properties, we can use the following modified pseudocode:

1: **function** MINIMAL-VALUE-ITERATION(mdp) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s'|s,a)$, rewards $R(s,a)$, discount γ
local variables: U,U', vectors of utilities for states in S
U_A, vector of utilities over states in S and actions in $A(s)$, initially R_{walk} for walk actions, R_{run} for run actions, and the terminal reward for exit actions in terminal states.
δ, the maximum change in the utility of any state in an iteration
C, C', sets of states whose U value changed on previous iteration
2: $\forall s \in S$, $U[s] \leftarrow \max_{a \in A(s)} R(s,a)$
3: $C' \leftarrow \{s \in S : U[s] \neq 0\}$
4: **repeat**
5: $U \leftarrow U'$; $\delta \leftarrow 0$; $C \leftarrow C'$; $C' \leftarrow \emptyset$
6: **for all** states $s \in S$ **if** $\exists a \in A(s), s' \in C$ such that $P(s'|s,a) > 0$ **do**
7: **for all** actions $a \in A(s)$ **if** $\exists s' \in C$ such that $P(s'|s,a) > 0$ **do**
8: $U_A[s,a] \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a) U[s']$
9: $U'[s] = \max_{a \in A(s)} U_A[s,a]$
10: **if** $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$, $C' \leftarrow C' \cup \{s\}$
11: **until** $\delta = 0$
12: **return** U

The savings are made in Line 6, where we consider only the states, s, for which there is some action, a, that might lead to a state, s', in our set of changed states, C. We use a similar restriction in our loop over actions (i.e., we ignore actions that lead to states whose values have not changed). Then we update our set of changed states in Line 10. Notice that even if $U_A[s,a]$ changes, $U[s]$ might not, because a might still be suboptimal. The utility updates themselves are unchanged from the original algorithm, so the changes will not alter the result. In other words, we are skipping only those steps that would not change the value of U anyway.

There are a few engineering shortcuts we can also use to make Line 10 faster. One is to leverage the fact that the transition probability, P, does not change, so we can precompute a reverse transition that, for each end state, s', returns the set of state-action combinations, s, a, that could lead to it. In GridWorld, for example, this function would return combinations like "(3,1),Move Down", "(1,2),Walk Left", etc. for state (1,1).

For policy iteration, we can make a similar observation as we do in value iteration. In particular, we loop over only the states, s, that can lead to other states, s', whose policy entry, $\pi(s')$, changed on the previous iteration.