In order to conclude that a graph is $k$-zombie win (or not) we must consider every possible zombie start and every possible survivor response.

Moreover, in the game described by Fitzpatrick et al., the zombies are somehow omniscient: whenever presented with a choice of shortest paths, they can choose the one which works to their advantage.

There's no way to know, a priori, which decision will lead to a zombie win and which will not. So in order to determine – programmatically – whether a graph is zombie win or not, we must not only explore all possible start configurations but also all possible zombie decisions (if any) encountered during the game.

Obviously we must also consider all possible survivor decisions, since it is possible that the survivor has not played optimally.

There do not seem to be any shortcuts or theoretical gadgets which could help us analyze complex graphs and so we are forced to examine all of these possibilities by brute force.

We present an algorithm which uses the minimax priniciple and implicit construction of graphs (cite Brassart & Bratley c.9) to explore all possible outcomes in a game. However, we must first discuss how to determine that a game is over.

## 1 Win Conditions

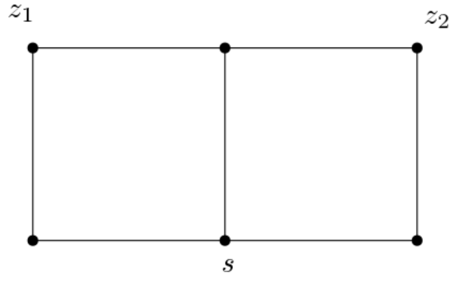
Obviously, if at any point a zombie occupies the same vertex as the survivor, then the survivor has lost. You could also stop one step earlier on the survivor's turn when the survivor is threatened and there is no escape.

But how do we know when the survivor has won? When playing the game on pen and paper, it becomes immediately clear when the survivor has won. Experienced players will even see it coming a few moves ahead. How to describe a survivor-win programmatically is less obvious.

**Lemma 1.** *If the game returns to a previously visited state and there were no opportunities for the zombies to play differently then the survivor has won.*

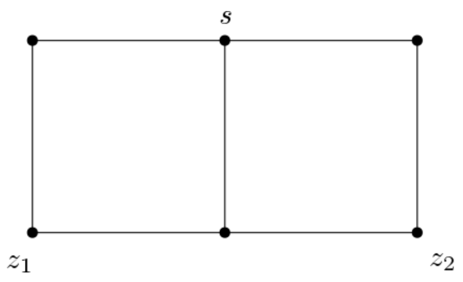
*Proof.* The survivor has found a sequence of moves which can be repeated infinitely and so has won by the rules of the game. □

However, revisiting a previous state is not sufficient to conclude that the survivor has won. It is quite possible that the zombies could have made different choices along the way and obtained a more favourable outcome. Consider the following simple graph and player positions:

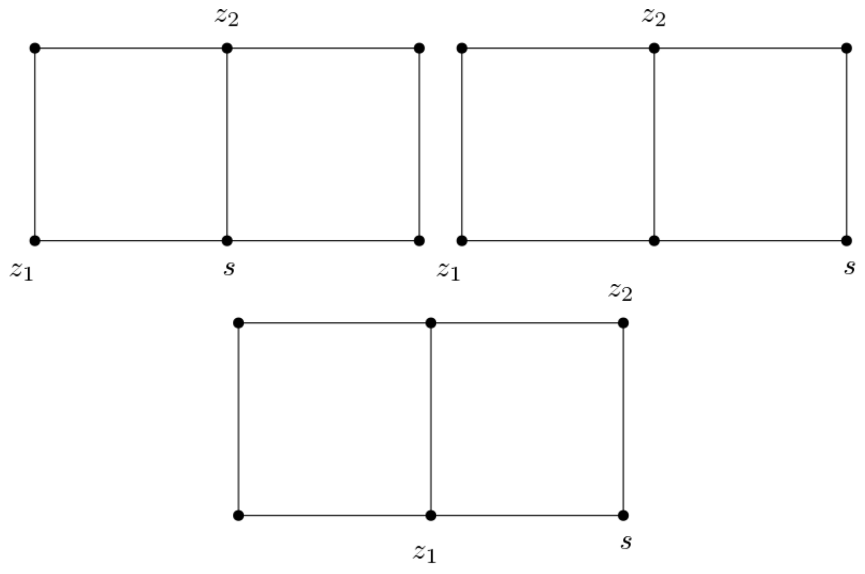


Both zombies have two choices available to them: move down or across. If the zombies are dumb, then they will both move across and the survivor has "won" so long as they continue to choose to move together.

Suppose now that both zombies move down and the survivor moves up.



On the next turn, the players could return to their previous positions. If the zombies keep making the same play, then they oscillate back and forth and again the survivor has "won". However, if the zombies play with any strategy then this game is zombie-win: one zombie moves down, and the other across, then the survivor is quickly encircled.



However, again we see that the zombies need to make the correct choice of shortest path.

So in order to conclude that a survivor has won, we need something stronger:

**Lemma 2.** *If there is a sequence of turns such that for any possible zombie-play we return to a previously visited state and the survivor is not captured, then the survivor has won.*

*Proof.* No matter what the zombies do, the survivor can repeat a sequence of moves without being captured and thus has won. □

We use this characterization of zombie-win and survivor-win in order to write a brute force algorithm for determining if a graph is $k$-zombie win.

## 2 Zombie-win brute force algorithm

Given a game graph $G = (V, E)$, we implicitly explore a directed graph $D = (V', A)$ which codifies the decisions made by each player and their outcomes. We assume that a Floyd-Warshall matrix of distances is available in global read-only memory.

A node of the digraph is a pair $(Z, s)$ where $Z = \{\text{the zombies' positions}\}$ is an ordered multiset and $s$ is the position of the survivor. We use a set rather than a vector because the zombies are all the same so the ordering does not matter.

Recall that we must consider all possible zombie-starts before concluding that a graph is survivor-win. For now, we fix a zombie start $Z \subset G$.

The root of the digraph is a zombie-start with $s$ unspecified. We draw an arc from the root to new nodes $(Z, s)$ for each $s \in V(G) \setminus N_G[Z]$. That is, for every safe survivor start. If there are no such arcs then the zombies dominate the graph and obviously have won.

Now we use minimax to test the outcome of each sequence of player decisions.

On the zombies' turn, we check if the survivor is caught ($s \in Z$) and, if so, return $-1$. Otherwise, we create nodes $(Z', s)$ for each possible combination of zombie moves from $Z$. These combinations are computed using the Floyd-Warshall matrix:

For each zombie $z_i$, probe $d(z_i, s)$ (constant time). Check the row $z_i$ of the matrix for each neighbour $v$ (which has distance 1) of $z_i$ (linear time). Test if $d(v, s) = d(z_i, s) - 1$. If so, then $v$ is a valid zombie move. Add $v$ to some set $Z_i$ of possible moves for $z_i$.

We obtain all possible combinations using the cartesian product of the $Z_i$ and return the minimum of all possible recursive calls to the survivor's move function with each $Z'$.

On the survivors' turn, we check to see if there is an ancestor with the same game state, or $(Z', s') = (Z, s)$. If so, then we know that this game state has already been visited and return 1.

Otherwise, we compute all possible survivor moves $S' = N_G[s] \setminus N_G[Z]$ and return the maximum of all recursive calls to the zombies' move function with each $s'$.

If the final return value is $-1$, then the zombies have won since there is a zombie start such that for any survivor start, the survivor is captured.

If the return value is 1, then the survivor has won since there exists a winning survivor strategy for every zombie start.

## 3 The Algorithm

```
Procedure zombieStart(G, k):
Require: Connected graph $G = (V, E)$ and $k \geq 1$
  $M \leftarrow$ Floyd-Warhsall distance matrix {Assume that $M$ is placed in global memory}
  result $\leftarrow 0$
  for all $Z = \{Z \subset V(G) \mid |Z| = k \wedge z_i \neq z_j \forall i \neq j\}$ do
    tmp $\leftarrow$ survivorStart($Z$)
    if result $= 0$ then
      result $\leftarrow$ tmp
    else if tmp $<$ result then
    result ← tmp
  end if
end for
return result
```

Procedure survivorStart(Z):

```
$S \leftarrow V(G) \setminus N_G[Z]$
if $S = \emptyset$ then
  return -1
end if
result ← 0
for all $s \in S$ do
  tmp ← zombieMove($Z$, $s$)
  if result = 0 then
    result ← tmp
  else if tmp > result then
    result ← tmp
  end if
end for
return result
```

Procedure zombieMove(Z, s):

```
for $i = 1$ to $k$ do
  $Z'_i \leftarrow \emptyset$
  $d \leftarrow M[z_i, s]$
  for $j = 1$ to $n$ do
    if $M[z_i, j] = 1 \wedge M[s, j] = d - 1$ then
      if $j = s$ then
        return -1
      end if{Zombie can capture survivor}
      $Z'_i \leftarrow Z'_i \cup \{j\}$
    end if
  end for
end for
result ← 0
for all Possible combinations $Z'$ from $\bigtimes_{i=1}^{k} Z'_i$ do
  tmp ← survivorMove($Z'$, $s$)
  if result = 0 then
    result ← tmp
  else if tmp < result then
    result ← tmp
  end if
end for
return result
```

Procedure survivorMove(Z, s):

```
if $Z, s$ is a previously visited state then
  return 1
end if{Implementing this line may be tricky and expensive. Will need to pass a list of previously visited states}
$S' \leftarrow N_G[s] \setminus N_G[Z]$
if $S' = \emptyset$ then
  return -1
```

```
end if
result ← 0
for all $s' \in S'$ do
  tmp ← zombieMove($Z$, $s'$)
  if result = 0 then
    result ← tmp
  else if tmp > result then
    result ← tmp
  end if
end for
return result
```