Turing Completeness and *Sid Meier's Civilization*

Adrian de Wynter

***Abstract*—We prove that three strategy video games from the *Sid Meier's Civilization* series: *Sid Meier's Civilization: Beyond Earth*, *Sid Meier's Civilization V*, and *Sid Meier's Civilization VI*, are Turing-complete. We achieve this by building three universal Turing machines—one for each game—using only the elements present in the games, and using their internal rules and mechanics as the transition function. We use these machines to prove the undecidability of these games under the unlimited turns, infinite map assumptions; and conclude by providing a sample execution of an algorithm—the three-state Busy Beaver—with one of our machines.**

***Index Terms*—Recreational mathematics, Turing completeness, *Sid Meier's Civilization*, strategy games.**

I. Introduction

The *Sid Meier's Civilization*[1] series is a collection of turn-based strategy video games, focused on building and expanding an empire through technological, social, and diplomatic development. The scale and scope of these games involve intricate rules, mechanics, and multiple victory conditions, which makes them hard to characterize in terms of their computational complexity.

Understanding the intrinsic difficulty of a game has implications on the choice of algorithms that can (and cannot) be developed to play and manage the game. This choice is in turn crucial to the user experience, for example, by designing adaptive difficulties [1], or effective combat strategies [2], [3]. For instance, Sudoku is known to be NP-complete [4], yet playable via optimization methods [5]. On the other hand, playing Go requires more sophisticated algorithms, such as residual networks [6], since it is known to be EXPTIME-complete under Japanese rules [7], [8]. In the extreme, Churchill *et al.* [9] proved that the card game *Magic: The Gathering* (MTG) is both Turing-complete and undecidable. Its undecidability means that there exist questions about this game that cannot be answered algorithmically. For instance, "does this MTG state have a sequence of winning moves?" cannot be answered.

In this article, we rigorously characterize the computational complexity of three installments of the *Civilization* series: *Sid Meier's Civilization: Beyond Earth* (Civ:BE from now on), *Sid Meier's Civilization V* (Civ:V), and *Sid Meier's Civilization VI* (Civ:VI). We prove that, under certain assumptions, these games are undecidable—that is, there does not exist an algorithm that plays the game perfectly. We carry out our analysis by showing that it is possible to build Universal Turing Machines (UTMs) inside running game sessions. These UTMs use only elements and rules present in the game. We conclude by providing an example implementation and execution of an algorithm—the three-state Busy Beaver game [10]—in a Turing machine built in Civ:BE.

Manuscript received 7 May 2021; revised 18 February 2022; accepted 6 April 2022. Date of publication 12 April 2022; date of current version 16 June 2023.

The author is with the The University of Texas at Austin, Austin, TX 78705 USA (e-mail: dewynter@utexas.edu).

Digital Object Identifier 10.1109/TG.2022.3166874

[1]All products, company names, brand names, trademarks, and images are properties of their respective owners.

II. Background

A. Turing Machines and Turing Completeness

A Turing machine can be visualized as an automaton that executes instructions as dictated by a function δ. These instructions involve reading or writing to a tape of infinite length via a head. Following the definition from Hopcroft *et al.* [11], Turing machines are a 7-tuple $\langle Q, \Gamma, \Sigma, \delta, b, q_0, F\rangle$, where:

1) Q is a set of possible *states* that can be taken internally by the Turing machine. Namely, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states.
2) Γ is the set of tape *symbols* on which the Turing machine can perform read/write operations. In particular, $b \in \Gamma$ is the blank symbol, and $\Sigma \subseteq \Gamma$, $b \notin \Sigma$ is the set of symbols which may be initially placed in the tape.
3) $\delta : (Q \setminus F) \times \Gamma \nrightarrow Q \times \Gamma \times \{L, R\}$ is the *transition function*. It is a partial function that takes in all states from the machine, along with the current read from the tape; it determines the next state to be taken by the Turing machine, as well as whether to move the head left (L) or right (R) along the tape. Not moving the tape can be trivially encoded in this definition.

Turing machines are mathematical objects that abstract out the notion of an algorithm. They are limited in practice due to the infinite length requirement on the tape. However, since they are are finite objects, it is possible to encode a Turing machine M_1 and use it as an input to a separate machine M_2, so that M_2 simulates M_1. A (m, n)-*universal Turing machine*, or (m, n)-UTM, is a Turing machine that is able to simulate any other Turing machine, and such that $|Q| = m, |\Gamma| = n$. The Church–Turing thesis states that the set of functions that can be computed by a (m, n)-UTM is precisely the set of computable functions [12], and hence they can be thought of as being as capable as any physical computer [13].

Any set of rules and objects that can be showed to simulate a UTM is said to be a *Turing-complete* model of computation. All Turing-complete models are equivalent to one another, in the sense that they can compute the same set of functions [14]. They encompass mathematical models such as the λ-calculus and the theory of μ-recursive functions; most programming languages; other rule-based systems like Conway's Game of Life [15] and MTG [9]; and video games such as *Minesweeper* [16] and software like Microsoft PowerPoint [17]. Designing a construct equivalent to a UTM requires assuming access to an unbounded resource (e.g., Minesweeper must have an infinite board, MTG must have access to unlimited tokens) to simulate the tape, as well as a way to keep track of the states the machine is in (e.g., the Game of Life uses dynamic patterns to manipulate portions of the grid).

B. General Mechanics of Sid Meier's: Civilization

All of the *Sid Meier's: Civilization* installments are turn-based, alternating, multiplayer games. The players are in charge of an empire,[2]

[2]In the case of Civ:BE, an extrasolar colony.

and good management of resources and relationships with other players are key to achieve one of the multiple possible victory conditions allowable. These victories (e.g., Diplomatic, Militaristic, Scientific) and their specific mechanics are of interest when analyzing the computational complexity of the game, but not when designing a Turing machine. In this article, we limit ourselves to describing the rules governing the actions of units on the map, and which are common to the games that we will be analyzing. Since the naming conventions for the components of the games vary across them—even when they serve the same purpose—we also introduce a unified notation.

The games that we will be discussing (Civ:BE, Civ:V, and Civ:VI) are played on a finite map divided in hexagonal tiles, or *hexes*. Hexes are composed of different geographical types (e.g., oceans, deserts, plains), and may contain certain features (e.g., forests, oil) dependent on the game's initial conditions.

All hexes have an inherent *resource* yield, which is used by the player to maintain their units and buildings, and to acquire more advanced *technologies* and *social policies*. Some examples of these resources are Food, Faith, Culture, and Science. One or more player-controlled units known as *Workers* are usually tasked to build *improvements* on a hex, which alters their yields. As an example, technologies are acquired by accruing Science, and Workers can build improvements on hexes (e.g., an Academy) to increase their Science yield.

With some notable exceptions, described in the following section, Workers may only build improvements in hexes owned by a *City*. A City can be seen as a one-hex special improvement, and has three main duties: to produce ("train") units, to control and expand the boundaries of the owner's frontiers, and to produce *Citizens* to work the hexes and their improvements. Citizens may only work the hexes owned by the City, but players are able to micromanage the placement of Citizens to alter their resource yield.

On every turn, the player executes all or some of the actions available to them, such as building improvements, researching technologies, and social policies; and focuses on the overall management of the empire. However, the specific mechanics around Cities, resources, features, available improvements, technologies, and social policies vary slightly from game to game. They are discussed in detail in the next section, as they are the key for building our UTMs.

III. Civ:BE, Civ:V, and Civ:VI Are Turing-Complete

For this section, the following two assumptions hold.

Assumption 1: The number of turns in a session of Civ:BE, Civ:V, or Civ:VI is infinite, and there is only one player in the game.

Assumption 2: The map in the session extends infinitely in either direction.

The rationale behind Assumption 1 is related to the fact that another player may interfere with the computation by—say—attacking the Workers, or reaching any of the victory conditions before the computation terminates. Note that Assumption 1 is attainable within the game itself. On the other hand, Assumption 2 is needed because we use the map as the tape in our proofs. It is as a theoretical requirement: assuming an unbounded resource for computation is inherent to the definition of a Turing machine, and hence common to all UTM constructions. However, arbitrarily large maps in any of the games are not attainable without the use of mods. Even then, the underlying hardware may run out of memory, thus rendering the construction impractical. We elaborate in Section V on how does that affect our results.

A. A $(10, 3)$-UTM in Civ:BE

Suppose we have two Workers carrying out instructions on separate, nonoverlapping sections of the map: one keeps track of states, and another acts as the head. The latter operates over the tape, which we consider to be the entire map minus the state hexes. We do not require the player to own any of the hexes on the tape. The state is tracked on nine, player-owned hexes. In Civ:BE, building any improvement takes a certain amount of turns, but the Workers may build and repair any number of them.

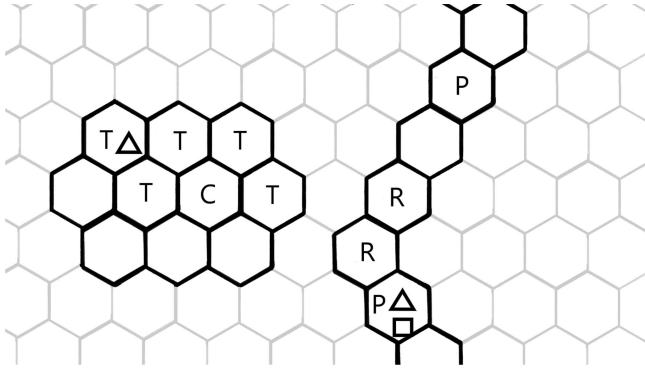

Fig. 1. Diagram of the $(10, 3)$-UTM from Theorem 1. The tape is located to the right of the image, and runs from top to bottom of the diagram. The head (the tape Worker and the Rover, denoted by a triangle and a square), is near the bottom, reading "Pillaged Road," denoted in the image with a P. Other symbols in the tape are "Road" (R) and "No Improvement" (blank). To the left, a City (C) has five Terrascapes (T) within its borders, and serves to track the states with the state worker (leftmost triangle). This machine is in state q_5: each Terrascape contributes C_* Culture, so $C_t = 5C_* + C_0$, and $(C_t - C_0)/C_* = 5$.

1) Tape: The tape is a continuous strip over the entire map, excluding the state hexes. Without loss of generality, we assume that it is comprised of flat, traversable, hexes. This is needed because irregular terrain requires more movement points to traverse. The set of symbols Γ is based off improvements that can be added and removed indefinitely by the tape Worker anywhere on the map—namely, Roads. The tape Worker adds and removes Roads according to the transition function, and a fast-moving unit (in this case, a Rover) pillages it. Note that for this setup to work, the Worker and the Rover must move at the same time. Then, $\Gamma = \{\text{No Improvement}, \text{Road}, \text{Pillaged Road}\}$.

2) The: Suppose the player owns at least nine flat, desert, unimproved hexes which will act as the states. Desert is required because we map states to the resource yields provided by a specific improvement—the *Terrascape*. Income from other hex types may interfere with state tracking. Moreover, we assume the player has unlocked the *civic* (social policy) *Ecoscaping*, which provides $+1$ Food, Production, and Culture for every Terrascape built, along with the relevant technology (*Terraforming*) required to build them. The Worker in charge of the states builds and remove Terrascapes on the state hexes, and the normalized change in Culture income serves as the state value $q_i \in Q$, $Q = \{0, \ldots, 9\}$. See Fig. 1 for a diagram of the UTM.[3]

Theorem 1: Suppose Assumptions 1 and 2 hold. Also suppose that there is a section of the map disjoint from the rest, and comprised of at least nine desert, unimproved, player-owned hexes; and that the player is able to build and maintain at least nine worked Terrascapes, each contributing C_* Culture per turn. If the player has a constant base (that is, minus the Terrascape yield) per-turn Culture C_0, and has the Ecoscaping civic, then the following is a $(10, 3)$-UTM:

$$Q = \{0, \ldots, 9\} \tag{1}$$

$$\Gamma = \{\text{No Improvement}, \text{Road}, \text{Pillaged Road}\} \tag{2}$$

[3]High-resolution images of all UTMs, built in running game sessions, can be found in https://adewynter.github.io/notes/CivUTMs.html.

TABLE I
TRANSITION FUNCTION FOR THE THREE-STATE TURING MACHINE CORRESPONDING TO BB-3, AND THE EQUIVALENT CIV:BE CONSTRUCTION

Civ:BE state (Δ; $tape\ read$)	Command to Workers ($tape$; $state$)	BB-3 TM
0; No Improvement	Build a Road and move R; Build a Terrascape	$q_0 0; 1Rq_1$
0; Road	Build a Road and move L; Build 2 Terrascapes	$q_0 1; 1Lq_2$
1; No Improvement	Build a Road and move L; Remove a Terrascape	$q_1 0; 1Lq_0$
1; Road	Build a Road and move R; No build	$q_1 1; 1Rq_1$
2; No Improvement	Build a Road and move L; Remove a Terrascape	$q_2 0; 1Lq_1$
2; Road	HALT	$q_2 1; 1R\ HALT$

For the Turing machine, the notation is read as ($state$, $read$; $write$, $move$, $set\ state$). For the Civ:BE machine, Δ corresponds to the normalized change in culture per turn.

TABLE II
BIJECTION BETWEEN THE UTM FROM THEOREM 1 AND ROGOZHIN'S $(10, 3)$-UTM

Civ:BE state (Δ; $tape\ read$)	Command to Workers ($tape$; $state$)	$(10,3)$-UTM
0; No Improvement	Build a Road and move R; No build	$q_0 0; 1Rq_0$
0; Road	Remove Improvement and move L; Build a Terrascape	$q_0 1; 0Lq_1$
0; Pillaged Road	No Improvement and move R; Build 3 Terrascapes	$q_0 b; bRq_3$
1; No Improvement	No Improvement and move L; Build a Terrascape	$q_1 0; 0Lq_2$
1; Road	Remove Improvement and move L; No build	$q_1 1; 0Lq_1$
1; Pillaged Road	No Improvement and move L; No build	$q_1 b; bLq_1$
2; No Improvement	No Improvement and move L; Remove a Terrascape	$q_2 0; 0Lq_1$
2; Road	Pillage the Road and move L; Build 3 Terrascapes	$q_2 1; bLq_5$
2; Pillaged Road	No Improvement and move R; Remove 2 Terrascapes	$q_2 b; bRq_0$
3; No Improvement	Build a Road and move R; Remove 3 Terrascapes	$q_3 0; 1Rq_0$
3; Road	No Improvement and move R; Build a Terrascape	$q_3 1; 1Rq_4$
3; Pillaged Road	Repair the Road and move L; No build	$q_3 b; 1Lq_3$
4; No Improvement	Build a Road, Pillage it, and move L; Remove 2 Terrascapes	$q_4 0; bLq_2$
4; Road	No Improvement and move R; No build	$q_4 1; 1Rq_4$
4; Pillaged Road	No Improvement and move R; No build	$q_4 b; bRq_4$
5; No Improvement	Build a Road and move L; Build a Terrascape	$q_5 0; 1Lq_6$
5; Road	No Improvement and move L; No build	$q_5 1; 1Lq_5$
5; Pillaged Road	No Improvement and move L; No build	$q_5 b; bLq_5$
6; No Improvement	No Improvement and move R; Build a Terrascape	$q_6 0; 0Rq_7$
6; Road	HALT	$q_6 1;\ HALT$
6; Pillaged Road	No Improvement and move L; Build 2 Terrascapes	$q_6 b; bLq_8$
7; No Improvement	Build a Road and move L; Remove 2 Terrascapes	$q_7 0; 1Lq_5$
7; Road	No Improvement and move R; No build	$q_7 1; 1Rq_7$
7; Pillaged Road	No Improvement and move R; No build	$q_7 b; bRq_7$
8; No Improvement	Build a Road and move L; Build a Terrascape	$q_8 0; 1Lq_9$
8; Road	Remove Improvement and move R; Build a Terrascape	$q_8 1; 0Rq_9$
8; Pillaged Road	Remove Improvement and move L; Remove 5 Terrascapes	$q_8 b; 0Lq_3$
9; No Improvement	No Improvement and move R; Remove 5 Terrascapes	$q_9 0; 0Rq_4$
9; Road	Remove Improvement and move R; No build	$q_9 1; 0Rq_9$
9; Pillaged Road	No Improvement and move R; Remove a Terrascape	$q_9 b; bRq_8$

For the Turing machine, the notation is read as ($state$, $read$; $write$, $move$, $set\ state$). For the Civ:BE machine, Δ corresponds to the change in culture per turn relative to the Player's base yield.

where $q_i \in Q$ is the difference in the total Culture C_t yields in turn t, $q_i = (C_t - C_0)/C_*$, and the transition function δ is as described in Table II .

Proof: A full proof can be found in Appendix A. It consists of two parts: building a bijection between δ and the transition function of an existing $(10, 3)$-UTM; and showing that our UTM is bounded. That is, Civ:BE's rules around movement, building, and other actions in Table II do not interfere with the execution time of any program in our machine.

The UTM from Theorem 1 is not the only possible UTM buildable within Civ:BE. For example, unlocking the ability of a Worker to add and remove Miasma expands the set of symbols from the tape, and yields a $(7, 4)$-UTM.

There are two things to highlight from this construction, and that apply to the rest of the UTMs presented in this article. First, although the base constant yield is hard to achieve in-game, it can be accounted for: simply execute every move at the beginning of the turn, record C_0, and then execute the transition function for the UTM. Counting the number of Terrascapes on a specific hex would work just as well.

Second, remark that this UTM, along with the others presented in the article, may be fully automated by using the API provided by the publisher. Manual simulation is, although tiresome, also possible, as shown in Section IV.

B. A $(10, 3)$-UTM in Civ:V

We follow the same approach as in Theorem 1: one Worker builds improvements (in this case, Roads and Railroads) on hexes to denote the symbols, and another Worker improves and unimproves hexes to encode the internal state of the machine. Just as in Civ:BE, building any improvement in Civ:V takes a certain amount of turns, but the Workers may build any number of them.

1) Tape: As in Theorem 1, the tape is a continuous strip over the entire map, and it is composed of flat, traversable, hexes. The set of symbols Γ is the two improvements that can be built by the tape Worker anywhere on the map: Roads and Railroads. Building a Railroad requires that the player has unlocked the *Engineering* technology. Then $\Gamma = \{\text{No Improvement}, \text{Road}, \text{Railroad}\}$.

TABLE III
BIJECTION BETWEEN THE UTM FROM THEOREM 2 AND ROGOZHIN'S $(10, 3)$-UTM

Civ:V state (Δ; $tape\ read$)	Command to Workers ($tape$; $state$)	$(10,3)$-UTM
0; No Improvement	Build a Road and move R; No build	q_0 ; 01, Rq_0
0; Road	Remove Improvement and move L; Build a Railroad	q_0 ; 10, Lq_1
0; Railroad	No Improvement and move R; Build 3 Railroads	q_0 ; bb, Rq_3
1; No Improvement	No Improvement and move L; Build a Railroad	q_1 ; 00, Lq_2
1; Road	Remove Improvement and move L; No build	q_1 ; 10, Lq_1
1; Railroad	No Improvement and move L; No build	q_1 ; bb, Lq_1
2; No Improvement	No Improvement and move L; Remove a Railroad	q_2 ; 00, Lq_1
2; Road	Build a Railroad and move L; Build 3 Railroads	q_2 ; $1b$, Lq_5
2; Railroad	No Improvement and move R; Remove 2 Railroads	q_2 ; bb, Rq_0
3; No Improvement	Build a Road and move R; Remove 3 Railroads	q_3 ; 01, Rq_0
3; Road	No Improvement and move R; Build a Railroad	q_3 ; 11, Rq_4
3; Magrail	Build a Road and move L; No build	q_3 ; $b1$, Lq_3
4; No Improvement	Build a Railroad and move L; Remove 2 Railroads	q_4 ; $0b$, Lq_2
4; Road	No Improvement and move R; No build	q_4 ; 11, Rq_4
4; Railroad	No Improvement and move R; No build	q_4 ; bb, Rq_4
5; No Improvement	Build a Road and move L; Build a Railroad	$q_5 0; 1Lq_6$
5; Road	No Improvement and move L; No build	$q_5 1; 1Lq_5$
5; Railroad	No Improvement and move L; No build	$q_5 b; bLq_5$
6; No Improvement	No Improvement and move R; Build a Railroad	$q_6 0; 0Rq_7$
6; Road	HALT	$q_6 1;\ HALT$
6; Railroad	No Improvement and move L; Build 2 Railroads	$q_6 b; bLq_8$
7; No Improvement	Build a Road and move L; Remove 2 Railroads	$q_7 0; 1Lq_5$
7; Road	No Improvement and move R; No build	$q_7 1; 1Rq_7$
7; Railroad	No Improvement and move R; No build	$q_7 b; bRq_7$
8; No Improvement	Build a Road and move L; Build a Railroad	$q_8 0; 1Lq_9$
8; Road	Remove Improvement and move R; Build a Railroad	$q_8 1; 0Rq_9$
8; Railroad	Remove Improvement and move L; Remove 5 Railroads	$q_8 b; 0Lq_3$
9; No Improvement	No Improvement and move R; Remove 5 Railroads	$q_9 0; 0Rq_4$
9; Road	Remove Improvement and move R; No build	$q_9 1; 0Rq_9$
9; Railroad	No Improvement and move R; Remove a Railroad	$q_9 b; bRq_8$

For the Turing machine, the notation is read as $(state,\ read;\ write,\ move,\ set\ state)$. For the Civ:v Machine, Δ corresponds to the number of railroads in the State Hexes.

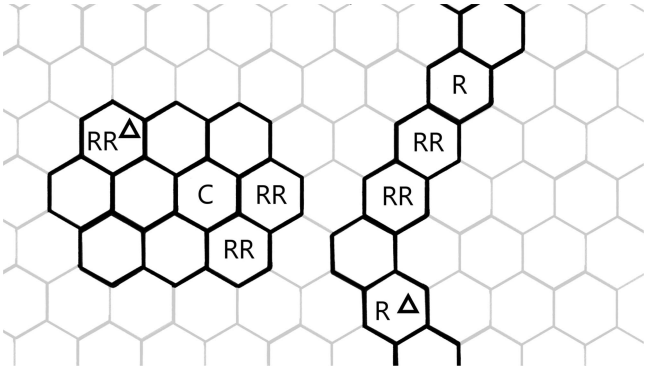

Fig. 2. Diagram of the $(10, 3)$-UTM from Theorem 2. It is in the state q_3 since the City (to the left, denoted by C) has three Railroads (RR) built within its borders. The tape, to the right, runs from top to bottom of the diagram. The head (rightmost triangle) is positioned over a hex with a Road (R).

2) States: In Civ:V it is no longer possible to remove improvements outside of Roads and Railroads, so we must rely on these improvements for state tracking. The Worker in charge of the states builds and removes Railroads to encode the state. This is carried out in a reserved section of the map (e.g., inside a City), and the total number of Railroads serves as the state value $q_i \in Q, Q = \{0, \ldots, 9\}$. See Fig. 2 for a diagram of the UTM.

Theorem 2: Suppose Assumptions 1 and 2 hold. Also suppose that there is a section of the map disjoint from the rest, owned by the player, and composed of at least nine hexes. Assume that the player has the Engineering technology.

Then the following is a $(10, 3)$-UTM:

$$Q = \{0, \ldots, 9\} \tag{3}$$

$$\Gamma = \{\text{No Improvement}, \text{Road}, \text{Railroad}\} \tag{4}$$

where $q_i \in Q$ is the total number of Railroads in the nine hexes, and the transition function δ is as described in Table III.

Proof: In Appendix B. It is analogous to the proof of Theorem 1.

Similar to Theorem 1, there are many viable UTM constructions in Civ:V. For example, it is possible to use a fast-moving pillaging unit (e.g., a Knight) and the ability of Workers to build Forts, to expand the symbol set of the machine to a $(5, 5)$-UTM.

C. A $(48, 2)$-UTM in Civ:VI

In Civ:VI, Worker units are unable to build improvements indefinitely. It is still possible to build an UTM in Civ:VI if we use the Cities as the head *and* the tape, and leave the Worker to encode only the position of the head. The Citizens are micromanaged to "write" symbols to the tape, by putting them to work specific hexes. As before, we maintain a separate set of hexes to act as the states.

In CiV:VI, there is no upper limit on the number of Cities founded by the player, as long as they are all placed at least three hexes away from other Cities. Under our current assumptions, this allows us to found an arbitrary number of Cities via a unit called a *Settler*. Settlers are units trained by Cities: once it has been created, the City in question loses a Citizen. With enough available Food and after a number of turns, a new Citizen will be born in the City. No Settlers can be trained if the City has only one Citizen.

Settlers can only be used once: founding a City consumes the Settler, and a City takes its place. The hexes owned at this time are set to be

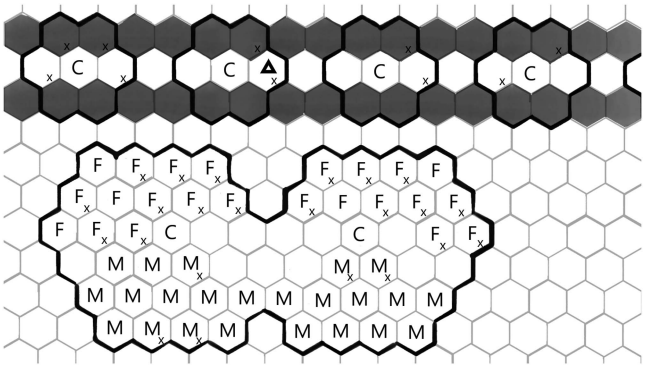

Fig. 3. Diagram of the $(48, 2)$-UTM from Theorem 3. Each City is denoted by C, and worked hexes are marked with an X. The bottom two Cities act as the state of the machine. The cities on the top strip contain each two cells from the tape. This machine is in q_{5n} since here are 5 Monasteries (M) being worked, and the head (above, as a triangle) is not at the end of the tape.

the six closest hexes,[4] and two Citizens spawn: one works the hex with the highest-possible yield of combined resources, and another—which cannot be retasked—the City itself. All of this occurs on the same turn.

We will utilize the available hexes and the Citizens to build our $(48, 2)$-UTM. Although our construction is more complicated than previous UTMs, we note that all of this is achievable with the built-in mechanics and rules of the game. A diagram of our construction is in Fig. 3.

1) Tape: Assume, for simplicity, that the map is composed of flat hexes, all of which are of the floodplains category. A single, uninterrupted, grassland strip is the only exception. The tape is then the grassland hexes owned by the player. Cities are collocated with at least three hexes between them over the grassland strip. The symbols on the tape, $\Gamma = \{\text{Is Being Worked, Is Not Being Worked}\}$, correspond to the number of Citizens working a given grassland hex.

Our design is unconventional since the tape is not infinite by design. Instead, we consider a transition to be concluded if the City has at least three Citizens. We cap the growth of any new City to four Citizens: one that works in the City (and cannot be moved), two to work the tape hexes, and one that will act as a Settler if need be. If a Settler has been created, then the growth of the City is capped at three. Note that this doubles as a marker for the end of the tape.

Every time the transition function issues a "move right" command, the controller first checks the population size of the City, and the position of the Worker unit. There are four possible cases as follows.

1) The City has three Citizens, and the Worker is on the rightmost (respectively, leftmost) hex. Then the Worker moves right (left), to the next City, and is placed on the leftmost (rightmost) hex.
2) The City has four Citizens. This is the end of the tape and a Settler must be trained to expand the tape. After it is spawned, the City is capped to three, and the Settler is issued the command to move right (respectively, left) four hexes, build a City, and grow it to four population. The Worker is issued the command to move right (left) three hexes, and wait until the City is grown.

The "move left" command is symmetric to the above.

2) States: Similar to Theorems 1 and 2, we rely on a set of hexes disjoint to the tape for our state tracking, and we utilize the relative yield of a specific resource (Faith) to encode it. We assume the player has built 23 Monasteries without any adjacent districts. Monasteries, when worked under these conditions, yield $+2$ Faith per turn, and are only buildable if the player has ever allied with the Armagh City-State. We also require the player to have built 23 Farms without any adjacent districts.[5] Transitioning from one state to the other requires the player to reassign as many workers as possible to adjust the difference in Faith yields from the beginning of time, which in turn realizes (part of) the state $q_i \in Q$.

Theorem 3: Suppose Assumptions 1 and 2 hold. Also suppose that the map contains an infinite strip of grassland hexes, with floodplains above and below, and that the player owns one City placed in the grassland strip. Also assume that there is a set of hexes disjoint to the strip, owned by the player, and with 23 Monasteries and 23 Farms without any adjacent districts. Let F_* be the constant base Faith yield per turn for this player.

Then, if there are no in-game natural disasters occurring during the time of the execution, the following Turing machine is a $(48, 2)$-UTM:

$$Q = \{0n, \ldots, 23n\} \cup \{0b, \ldots, 23b\} \tag{5}$$

$$\Gamma = \{\text{Is Being Worked, Is Not Being Worked}\} \tag{6}$$

where $q_{ix} \in Q$ is the difference in the Faith yields at turn t, $q_i = F_t - F_*$; $x \in \{n, b\}$ indicates whether it is needed to build a new Settler, and the transition function δ is as described in Table IV.

Proof: In Appendix C. Similar to Theorems 1 and 2, but we also show that the tape-expanding scheme for our UTM is feasible under the game's mechanics.

Corollary 1: Civ:BE, Civ:V, and Civ:VI are undecidable under Assumptions 1 and 2.

Proof: By using the results from Theorems 1, 2, and 3. Since the games are Turing-complete, any set of actions A that can be taken in any of the games has a correspondent UTM set of transitions. Then the question "is there an algorithm that decides whether an arbitrary A and a game state s guarantee victory?" is undecidable, by reduction to the Halting problem. A more detailed proof can be found in Appendix D.

We discuss the feasibility of Corollary 1 during normal play in Section V. Prior to that, we demonstrate a sample execution of an arbitrary algorithm within a running session of Civ:BE, by using our Turing machine.

IV. Example: The Busy Beaver in Civ:BE

The Busy Beaver game, as described by Radó [10], is a game in theoretical computer science. It challenges the "player" (a Turing machine or a person) to find a halting Turing machine with a specified number of states n, such that it writes the most number of 1 s in the tape out of all possible n-state Turing machines. The winner is referred to as the nth Busy Beaver, or BB-n. Note that all the Turing machines playing the Busy Beaver game have, by definition, only two symbols, $\Gamma = \{0, 1\}$.

Finding whether a given Turing machine is a Busy Beaver is undecidable, but small Busy Beavers are used to demonstrate how a Turing machine works. In particular, BB-3 is a three-state ($Q = \{q_0, q_1, q_2\}$), two-symbol ($\Gamma = \{0, 1\}$) Turing machine with a transition function as described in Table I.

Given the Turing-completeness of Civ:BE, we can build BB-3 within the game itself. An equivalent construction with our Civ:BE $(10, 3)$-UTM has the states $Q = \{0, 1, 2\}$, symbols $\Gamma =$

[4]This action technically builds a City Center, but we will forgo this distinction. Players who play as Russia will found Cities with five extra hexes.

[5]As in the other constructions, we impose specific requirements for our proofs and their generalizability. In practice, any resource yield (e.g., Culture) or improvement (e.g., Ethiopia's Rock-Hewn Church; Egypt's Sphinx; India's Stepwell) would work.

TABLE IV
RELATION BETWEEN THE UTM FROM THEOREM 3 AND ROGOZHIN'S $(24, 2)$-UTM, FOR THE STATES NOT REQUIRING A NEW SETTLER

Civ:VI state (Δ; $tape\ read$)	Command to Workers ($tape$; $state$)	$(24, 2)$-UTM
0; Is Not Being Worked	Is Not Being Worked, move R; Work 5 more Monasteries	$q_1 0; 0Rq_5$
0; Is Being Worked	Is Being Worked, move R; Work 2 more Monasteries	$q_1 1; 1Rq_2$
1; Is Not Being Worked	Is Being Worked, move R; Work 0 more Monasteries	$q_2 0; 1Rq_1$
1; Is Being Worked	Is Being Worked, move L; Work 2 more Monasteries	$q_2 1; 1Lq_3$
2; Is Not Being Worked	Is Not Being Worked, move L; Work 2 more Monasteries	$q_3 0; 0Lq_4$
2; Is Being Worked	Is Not Being Worked, move L; Work 0 more Monasteries	$q_3 1; 0Lq_2$
3; Is Not Being Worked	Is Being Worked, move L; Work 9 more Monasteries	$q_4 0; 1Lq_{12}$
3; Is Being Worked	Is Not Being Worked, move L; Work 6 more Monasteries	$q_4 1; 0Lq_9$
4; Is Not Being Worked	Is Being Worked, move R; Work 3 more Farms	$q_5 0; 1Rq_1$
4; Is Being Worked	Is Not Being Worked, move L; Work 2 more Monasteries	$q_5 1; 0Lq_6$
5; Is Not Being Worked	Is Not Being Worked, move L; Work 2 more Monasteries	$q_6 0; 0Lq_7$
5; Is Being Worked	Is Being Worked, move L; Work 2 more Monasteries	$q_6 1; 1Lq_7$
6; Is Not Being Worked	Is Being Worked, move L; Work 2 more Monasteries	$q_7 0; 1Lq_8$
6; Is Being Worked	Is Not Being Worked, move L; Work 0 more Monasteries	$q_7 1; 0Lq_6$
7; Is Not Being Worked	Is Not Being Worked, move L; Work 0 more Monasteries	$q_8 0; 0Lq_7$
7; Is Being Worked	Is Being Worked, move R; Work 5 more Farms	$q_8 1; 1Rq_2$
8; Is Not Being Worked	Is Not Being Worked, move R; Work 11 more Monasteries	$q_9 0; 0Rq_{19}$
8; Is Being Worked	Is Being Worked, move L; Work 4 more Farms	$q_9 1; 1Lq_4$
9; Is Not Being Worked	Is Being Worked, move L; Work 5 more Farms	$q_{10} 0; 1Lq_4$
9; Is Being Worked	Is Not Being Worked, move R; Work 4 more Monasteries	$q_{10} 1; 0Rq_{13}$
10;Is Not Being Worked	Is Not Being Worked, move L; Work 6 more Farms	$q_{11} 0; 0Lq_4$
10;Is Being Worked	HALT	$q_{11} 1;\ HALT$
11;Is Not Being Worked	Is Not Being Worked, move R; Work 8 more Monasteries	$q_{12} 0; 0Rq_{19}$
11;Is Being Worked	Is Being Worked, move L; Work 3 more Monasteries	$q_{12} 1; 1Lq_{14}$
12; Is Not Being Worked	Is Not Being Worked, move R; Work 2 more Farms	$q_{13} 0; 0Rq_{10}$
12; Is Being Worked	Is Being Worked, move R; Work 12 more Monasteries	$q_{13} 1; 1Rq_{24}$
13; Is Not Being Worked	Is Not Being Worked, move L; Work 2 more Monasteries	$q_{14} 0; 0Lq_{15}$
13; Is Being Worked	Is Being Worked, move L; Work 2 more Farms	$q_{14} 1; 1Lq_{11}$
14; Is Not Being Worked	Is Not Being Worked, move R; Work 2 more Monasteries	$q_{15} 0; 0Rq_{16}$
14; Is Being Worked	Is Being Worked, move R; Work 3 more Monasteries	$q_{15} 1; 1Rq_{17}$
15; Is Not Being Worked	Is Not Being Worked, move R; Work 0 more Monasteries	$q_{16} 0; 0Rq_{15}$
15; Is Being Worked	Is Being Worked, move R; Work 5 more Farms	$q_{16} 1; 1Rq_{10}$
16; Is Not Being Worked	Is Not Being Worked, move R; Work 0 more Monasteries	$q_{17} 0; 0Rq_{16}$
16; Is Being Worked	Is Being Worked, move R; Work 5 more Monasteries	$q_{17} 1; 1Rq_{21}$
17; Is Not Being Worked	Is Not Being Worked, move R; Work 2 more Monasteries	$q_{18} 0; 0Rq_{19}$
17; Is Being Worked	Is Being Worked, move R; Work 3 more Monasteries	$q_{18} 1; 1Rq_{20}$
18; Is Not Being Worked	Is Being Worked, move L; Work 15 more Farms	$q_{19} 0; 1Lq_3$
18; Is Being Worked	Is Being Worked, move R; Work 0 more Monasteries	$q_{19} 1; 1Rq_{18}$
19; Is Not Being Worked	Is Being Worked, move R; Work 1 more Farms	$q_{20} 0; 1Rq_{18}$
19; Is Being Worked	Is Not Being Worked, move R; Work 1 more Farms	$q_{20} 1; 0Rq_{18}$
20; Is Not Being Worked	Is Not Being Worked, move R; Work 2 more Monasteries	$q_{21} 0; 0Rq_{22}$
20; Is Being Worked	Is Being Worked, move R; Work 3 more Monasteries	$q_{21} 1; 1Rq_{23}$
21; Is Not Being Worked	Is Being Worked, move L; Work 1 more Monasteries	$q_{22} 0; 1Lq_{10}$
21; Is Being Worked	Is Being Worked, move R; Work 0 more Monasteries	$q_{22} 1; 1Rq_{21}$
22; Is Not Being Worked	Is Being Worked, move R; Work 1 more Farms	$q_{23} 0; 1Rq_{21}$
22; Is Being Worked	Is Not Being Worked, move R; Work 1 more Farms	$q_{23} 1; 0Rq_{21}$
23; Is Not Being Worked	Is Not Being Worked, move R; Work 10 more Farms	$q_{24} 0; 0Rq_{13}$
23; Is Being Worked	Is Not Being Worked, move L; Work 20 more Farms	$q_{24} 1; 0Lq_3$

For the $(24, 2)$-UTM, the notation is read as $(state,\ read;\ write,\ move,\ set\ state)$. For the Civ:VI machine, Δ is the change in faith from the Player's base yield, and the command "work n Monasteries/Farms" involves reassigning citizens to farms or monasteries.

$\{$No Improvement, Road$\}$, and with a transition function as described in Table I.[6]

V. DISCUSSION

We introduced UTMs for Civ:BE, Civ:V, and Civ:VI, and illustrated their ability to execute an arbitrary algorithm by running BB-3 on our Civ:BE UTM. We also proved that these games are undecidable under the unbounded turn limit and infinite map size assumptions.

An infinitely large map may seem like a largely theoretical construct, effectively limiting the practical implementation of any Turing machine by the hardware. This, however, is not a concern: the set of possible configurations can be expanded arbitrarily with a small overhead [11]. For example, a computer can prompt the user to add more memory when needed; and the MTG [9] and Civ:VI UTMs contain tape-expansion schemes as part of their construction. Circumventing this limitation in our UTMs could be as simple as breaking down the tape into multiple game saves. The final implementation will be more complex, although certainly realizable [11].

From Corollary 1, it follows that Civ:BE, Civ:V, and Civ:VI are, in principle, among the hardest games in the world. The key insight to show this was that a set of moves in the game could be reduced to a smaller set of UTM moves in the same game. Hence the question "is there a winning set of moves for an arbitrary state of the game?"

[6]An animated version of this execution can be found in https://adewynter.github.io/notes/CivBB3.html.

is undecidable. As Hearn and Demaine [8] point out, the existence of undecidable games is interesting in its own right: while still able to perform universal computation, they do so on the bounded-resource setting.

Our result only holds under the unbounded time and space assumptions. We do not expect this result to be true in bounded-resource strategic play. However, we conjecture that under the limited-turns constraint and arbitrary map size, the games are in EXPTIME. If true, this would imply the existence of algorithms analogous to the ones for chess or Go that can play the game effectively, by adapting to the player's actions–as of now, it relies on having advantages over the player [18]. Further work characterizing these and other games in terms of their complexity class could provide information on the type of algorithms that can be used to play these games perfectly, or, at least, more efficiently. As discussed in Section I, the right balance between the complexity and difficulty of a game is key for an engaging and rewarding experience.

Appendix A
Proof of Theorem 1

In this section, we prove Theorem 1. We show that there exists a bijection (see Table II) between the transition functions of our Civ:BE UTM and the $(10, 3)$-UTM program from Rogozhin [19]. Similar proofs for other models of computation can be found in [9], [20]–[23].

We show boundedness by evaluating the maximum number of turns required for any transition. Namely, the execution of every command takes at most the time taken to build (or remove) five Terrascapes and one Road. Let T and M the number of turns needed to build a Terrascape and a Road, respectively. Repairing and removing improvements takes only one turn. Then the maximum overhead, for any instruction, will be at most $5\,T + M$ turns. Since T and M are constants set ahead of time, and dependent on the game speed, it follows that our UTM does not incur a penalty in execution due to the game's mechanics.

Appendix B
Proof of Theorem 2

Table III shows a bijection between the transition functions of our Civ:V UTM and the $(10, 3)$-UTM program from Rogozhin [19]. We show boundedness as in Appendix A. The execution of every command will take at most the time taken to build (or remove) five Railroads, intion to removing a Railroad and building a Road. This is because the mechanics of Civ:V do not allow the Workers to build a Road on a Railroad hex, so the Railroad must be removed first. This last action takes one turn. Let $B_{rr} > 1$ be the cost of building a Railroad. The state Worker will have to build at most 3 Railroads; and the tape Worker will have to—at most—remove a Road and build a Railroad instead, at a cost of $1 + B_{rr}$ turns. Then the maximum overhead, for any instruction, is at most $4B_{rr} + 1$ turns. Since B_{rr} is a constant set ahead of time, and dependent on the game speed, it follows that this construction does not incur a penalty in execution due to the game's mechanics.

Appendix C
Proof of Theorem 3

In this section, we present a partial proof of Theorem 3: Table IV shows a bijective function between the transition function of our Civ:VI UTM and Rogozhin's $(24, 2)$-UTM. The mechanics around expanding cities are left out, as it is understood that any R or L command would have an extra branching statement to test the population size and train the Settler, making Table IV a $(48, 2)$-UTM. Concretely, let s_t, s'_t be states of the machine at some turn t, with $n \in s_t$ and $b \in s'_t$ being the only distinction between them. Then s_t is equivalent to: executing s_t, training a Settler, founding a City, growing it to 4 population, and then executing s_{t+1}. On the other hand, s'_t would immediately execute s_{t+1}.

Maintaining a City as an operational unit of the tape is feasible: each Citizen in a City consumes 2 Food per turn, so at any point in time the City will need at most 8 Food per turn. The City itself has a base yield of 2 (+2 since it is on grassland), and both grassland hexes being worked at the same time will provide 4 Food. This totals a maximum 8 Food per turn. It can be seen that this is also the minimum possible amount of Food produced by the City, since floodplains provide +3 Food. Hence our tape-expanding scheme is allowed under Civ:VI's mechanics.

Let L be the length of the tape at any given time. Then the number of Cities in the tape is $L/2$, and the cost of training a new Settler (in Production units) will be $15\,L + 50$, or $S(L)$ turns. Let C be the number of turns required to grow a City to 8 population. It takes a Settler 3 turns to found a new City. Then the total overhead for this UTM is given by $C + S(L) + 3$. Although C is a constant, but $S(L)$ depends on the length of the tape. Thus this UTM incurs a linear penalty in execution due to the game's mechanics—acceptable for a UTM due to their polynomial delay in simulating a Turing machine [14].

Appendix D
Proof of Corollary 1

Choose any of Civ:BE, Civ:V, or Civ:VI. Let $U = \langle Q, \Gamma, \Sigma, \delta, b, q_0, F\rangle$ be its corresponding UTM. Consider the case where a player desires to know, given an arbitrary state s of the game, whether there exists a sequence of actions (moves) $A = \{a_1, a_2, \dots\}$ that guarantees victory. These moves belong to the set of all possible legal moves executable by the player in the game.

Every $a_i \in A$ has a corresponding representation as a sequence of transitions T from an arbitrary UTM U'.

By Turing-completeness, U is equivalent to U', and T is expressible in terms of a sequence of transitions of U; hence every $a_i \in A$ has an equivalent encoding as a set of in-game legal moves and states belonging to Q and Γ.

It follows that A can be expressed in terms of a separate Turing machine $M_A = \langle Q_A, \Gamma_A, \Sigma_A, \delta_A, b_A, q_{0,A}, F_A\rangle$, for $Q_A \subset Q, \Gamma_A \subset \Gamma, \Sigma_A \subset \Sigma$, and, without loss of generality, $\delta_A = \delta, b_A = b, F_A = F, q_{0,A} = q_0$. Hence, this question is analogous to asking whether there is an algorithm that decides whether M_A halts given s. For arbitrary M_A and s, there does not exist such an algorithm: it is the Halting problem, and it is undecidable [11], [13], [14].

Acknowledgment

The author would like to thank the editors and anonymous reviewers, whose comments improved the quality of this article.

References

[1] M. Hendrix, T. Bellamy-Wood, S. McKay, V. Bloom, and I. Dunwell, "Implementing adaptive game difficulty balancing in serious games," *IEEE Trans. Games*, vol. 11, no. 4, pp. 320–327, Dec. 2019.

[2] A. Uriarte and S. Ontañón, "Combat models for RTS games," *IEEE Trans. Games*, vol. 10, no. 1, pp. 29–41, Mar. 2018.

[3] H. M. Wang, C.-Y. Hou, and C.-T. Sun, "Using simple design features to recapture the essence of real-time strategy games," *IEEE Trans. Games*, to be published, doi: 10.1109/TG.2021.3128753.

[4] T. Yato and T. Seta, "Complexity and completeness of finding another solution and its application to puzzles," *ICEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. E86-A, no. 5, pp. 1052–1060, 2003.

[5] H. Lloyd and M. Amos, "Solving Sudoku with ant colony optimization," *IEEE Trans. Games*, vol. 12, no. 3, pp. 302–311, Sep. 2020.
[6] T. Cazenave, "Residual networks for computer Go," *IEEE Trans. Games*, vol. 10, no. 1, pp. 107–110, Mar. 2018.
[7] J. M. Robson, "The complexity of Go," in *Proc. 9th World Comput. Congr. Inf. Process.*, 1983, pp. 413–417.
[8] R. A. Hearn and E. D. Demaine, *Games, Puzzles, and Computation*. Boca Raton, FL, USA: A. K. Peters/CRC Press, 2009.
[9] A. Churchill, S. Biderman, and A. Herrick, "Magic: The Gathering is Turing complete," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1904.09828
[10] T. Radó, "On non-computable functions," *Bell Syst. Tech. J.*, vol. 41, no. 3, pp. 877–884, 1962.
[11] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Upper Saddle River, NJ, USA: Pearson, 2013.
[12] H. Rogers Jr, *The Theory of Recursive Functions and Effective Computability*. Cambridge, MA, USA: MIT Press, 1987.
[13] M. Sipser, *Introduction to the Theory of Computation*. Cengage Learning, 2012, ch. 3.
[14] S. Arora and B. Barak, *Computational Complexity: A. Modern Approach*. Cambridge, U.K.: Cambridge Univ. Press, 2016.
[15] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for Your Mathematical Plays*. Wellesley, MA, USA: A. K. Peters Ltd., 2004.
[16] R. Kaye, "Infinite versions of Minesweeper are Turing complete," Accessed: Apr. 20, 2021. [Online]. Available: http://web.mat.bham.ac.uk/R.W.Kaye/minesw/infmsw.pdf
[17] T. Wildenhain, "On the Turing completeness of MS PowerPoint," Accessed: Apr. 20, 2021. [Online]. Available: https://www.andrew.cmu.edu/user/twildenh/PowerPointTM/Paper.pdf
[18] "Difficulty level (Civ6)," Accessed: Feb. 16, 2022. [Online]. Available: https://civilization.fandom.com/wiki/Difficulty_level_(Civ6)
[19] Y. Rogozhin, "Small universal Turing machines," *Theor. Comput. Sci.*, vol. 168, no. 2, pp. 215–240, 1996.
[20] M. Cook, "Universality in elementary cellular automata," *Complex Syst.*, vol. 15, pp. 1–40, 2004.
[21] R. Gorrieri, "CCS(25,12) is Turing-complete," *Fundamenta Informaticae*, vol. 154, no. 1–4, pp. 145–166, 2017.
[22] A. Smith, "Universality of Wolfram's (2,3) Turing machine," *Complex Syst.*, vol. 29, no. 1, pp. 1–44, 2020. [Online]. Available: https://doi.org/10.25088/ComplexSystems.29.1.1
[23] X. Zhang, L. Pan, and A. Păun, "On the universality of axon P systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2816–2829, Nov. 2015.